30/05/22

# Testing

- Manual Testing
- Automation Testing
  - (1) Java + (2) Selenium

company

step-I
1) Devolp the s/w or application → step-II
2) To test the devolped app or software.

# Java :- It is handled by orcal company devolper.

- It is programming language which is used to devolp the software.

**Software :-** It is a collection of programms which takes user input and produces desired output.

Program - it is set of instructions. to perform the task.

**For ex:** software / APP

Flipkart
- → login ⇒ { } -. Program is set
- → order ⇒ { == } - -11 ——
- → Payment ⇒ { =_ } - Progeam -11 ——
- → logout ⇒ { -_ } → Peogram -11 ——

All programs are in Java

**software / Application**

Flipkart
- → login ⇒ For login their is programm
- → order placed ⇒ -11 —— programm
- → Payment ⇒ { -_ } ⇒ programm
- → Logout ⇒ { -= . } peogramm

Features of Java :- (5 features)

(1) It is simple language. [Java provides inbuilt libraries, inside the inbuilt labraries their are programms already, so we can copy paste that programms and we can make an good application or we can make good software]

(2) It is Freely available. [we can this language without paying any single rupee]

(3) It is a open source.
[we can see the source code]

## Source Code :-

↳ Source code is the language or string of words, numbers, letters and symbols that a computer programm uses.

↳ It is a source of computer program. It contains declaration, instructions, functions, loops and other statements, which act as instructions for the programm on how to function.

Questions :-

1) which programming language u r using for automation testing?
2) what is mean by slw?
3) what is program?
4) what are the features of Java programming language?

31/05/22

(4) It is secure language.
(5) It is platform independent.

* Platform

* why it is secure language?
↳ Java is secure language [since java c converts the source file into the .classfile as the .classfile can't be understandable by the system/user. Hence it is a secure language]

* Why it is platform independent?
↳ It is platform independent [The program which is written in one operating system can easily run in other operating system. Hence it is called as platform independent]

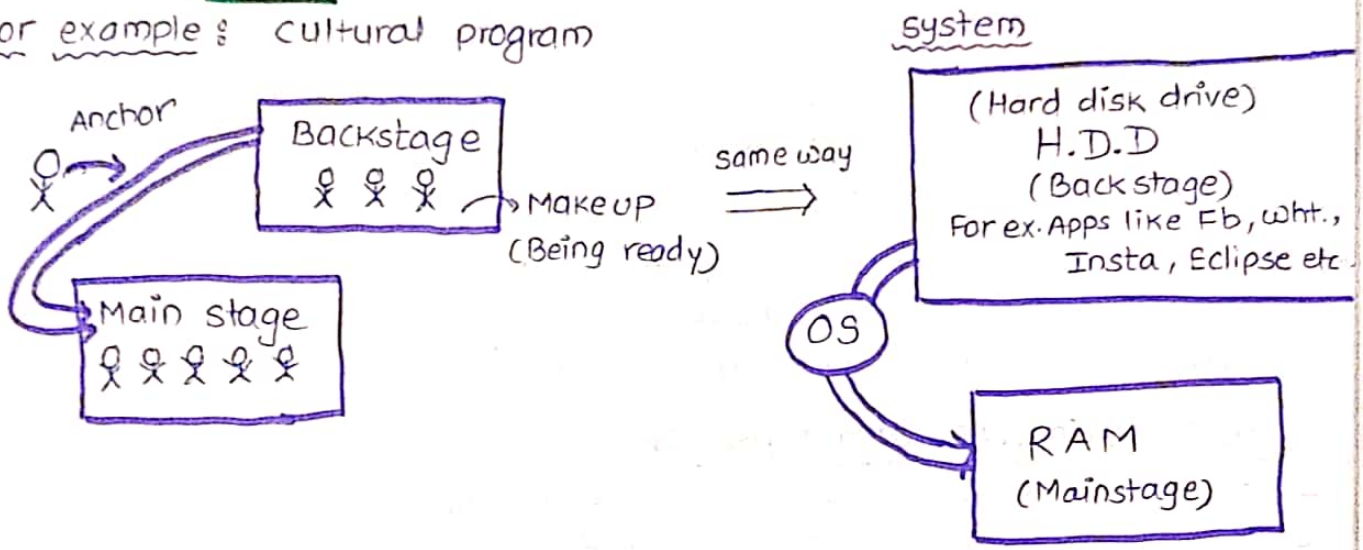
For e.g. We have operating system likes windows, Ubantu, Linux, macos etc.

Q) Why operating system is called as platform?

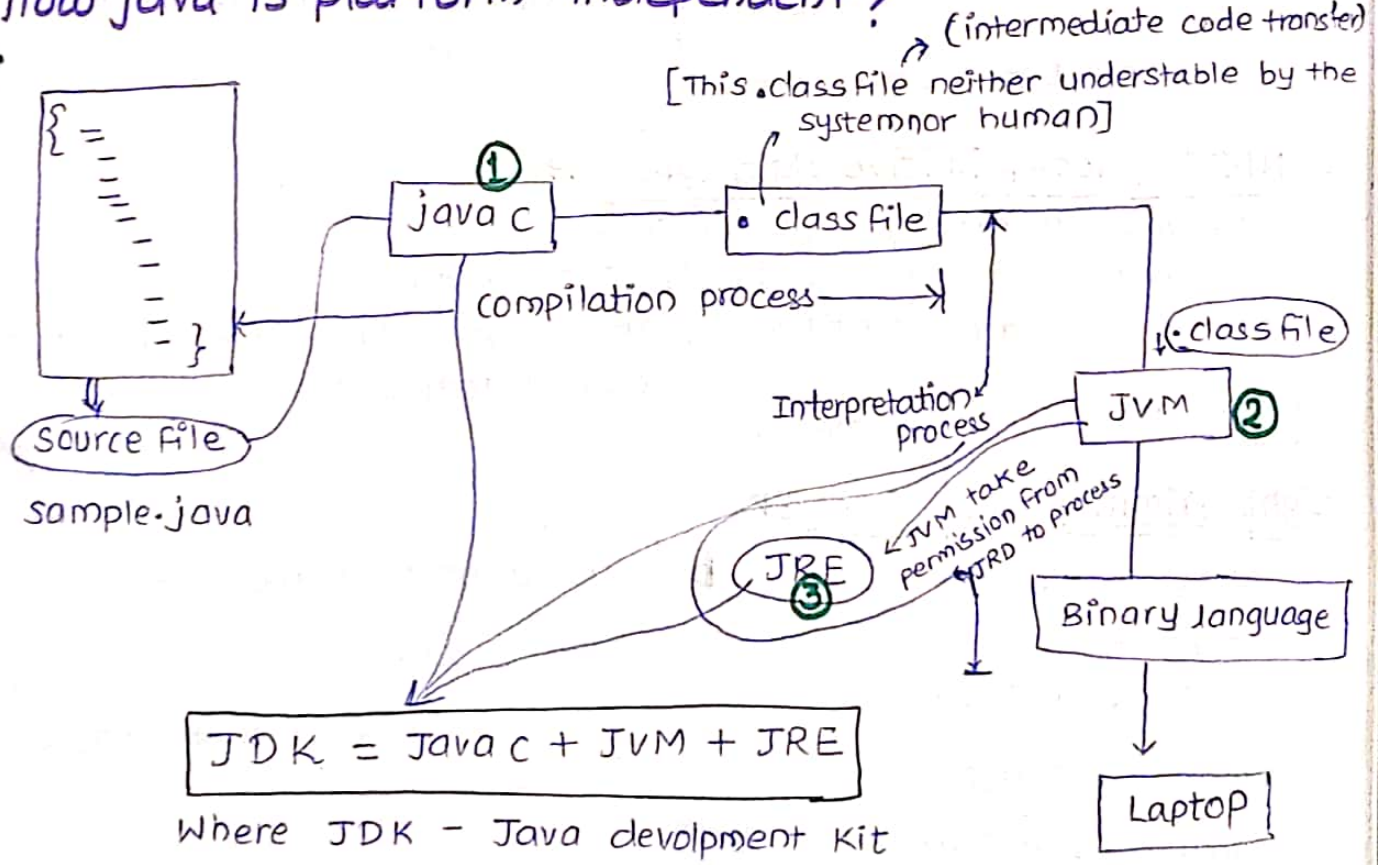
↳ Operating system (OS) is responsible to run other applications in HDD, Hence OS is called as platform.
- OS like windows, Linux, Macos, Ubantu etc.

## Platform (OS)

For example: cultural program




## * How java is platform independent?




JDK = Java c + JVM + JRE

Where JDK - Java devolpment Kit

Java c - Java compiler

JVM - Java virtual machine

JRE - Java Runtime enviorement.

01/06/22

# Data type :- The small portion of information is called as Data.

Ex :- she is Rita → Total is ~~sender~~ into. but rita is fact.
↳ Data

Data :- The actual fact which describes the attributes of entity is called as Data.

Types of Data :- (i) It is used to describe which type of data we are going to store.
(ii) It will give size of that particular data.

↳ There are two types of data types in java.
(1) PDT (Primitive data type) ⇒ (8 types)
(2) NPDT (Non-primitive data type) ⇒ (Any no. of types) (infinite)

## (1) PDT (Primitive data types) :-

- There are 8 types of primitive data type.
- The memory size of PDT is fixed.
- All PDT's are keywords (They are always started with the small letters or lower case)

Ex : int, byte, long etc.

## (2) NPDT (Non-primitive data types) :-

- There are infinite no. of NPDT's.
- The memory size of NPDT is not fixed.
- All NPDTS are identifiers (They are always started with the capital letters or upper case)

Ex :- String, Classname etc.

## Eight Primitive Data Types :-

(1) byte → 1 byte
(2) short → 2 byte
(3) int → 4 byte
(4) long → 8 byte
(5) double → 8 byte
(6) float → 4 byte
(7) char → 2 byte
(8) boolean → 1-bit

- byte is a unit of memory

1 byte = 8-bits

Ex : 1 0 0 0 1 1 1 1 ⇒ 1-byte
8 7 6 5 4 3 2 1
single bit

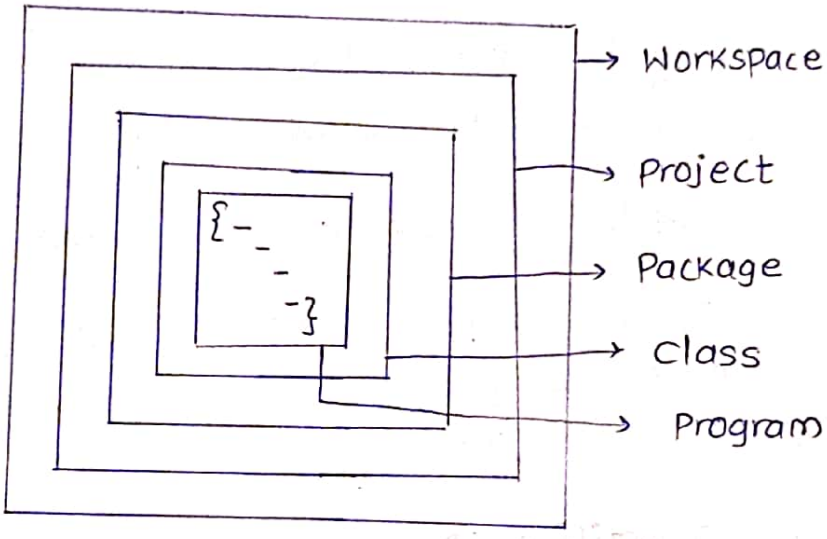
# Installation of Eclipse :-

**Eclipse :-** It is a application in which we can write Java programs

**STEPS TO Install :**

1. Go in google and search for Eclipse download for windows 64 bit.
2. click on downloads.
3. Extract downloaded file (Right click on downloaded file and click on extract all).

# FLOW :-




(Process to create Project and Package)

→ Project

Kiran ⟸ double click on it

⇓

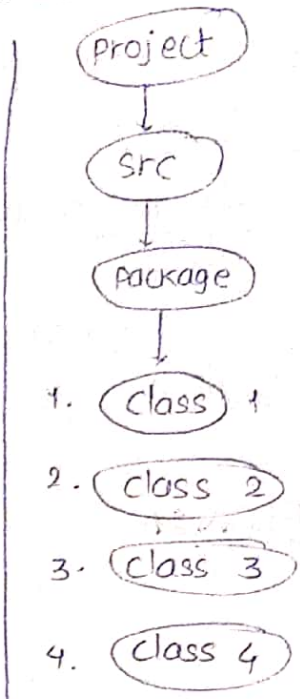
src ← Right click on src ⇒ New (cursor move towards right) → then

click on →

| Java project |
|---|
| Project |
| ENUM |
| Package |
| . . . . |

## Structure of Java Program :⇒

```
Package Kiran;
Class Sample { // class Body open
    main (-) { // main method body open
        ____________;
        ____________;
        ____________;
        ____________;
    } // main method body close
} // class Body closed
```




- Each and every program starts with main method whose system is

```
Public static void main ( String [] args) {
```

shortcuts : (i) main (control + space)
(ii) syso ( control + space)

## # Keywords and Identifiers :⇒

1. Keywords :⇒ These are the predefined words in java which has specific meaning, in which the devolper or tester can not change the meaning of keywords.

Ex: extends, break, implements, void, static etc.

- The keywords always starts with lower case / small letters.

2. Identifiers :⇒ These are the names given to classes, packages, methods, interfaces, Enums, variables etc.

- To write the identifiers there are some rules (Mandatory).
- And According to Java standard practice we have to follow some conventions also.

Rules for Identifiers :⇒

(i) Keywords can not be identifier.

(ii) It can not accept special symbols, any spaces except _ and $
↳ (underscore symbol)

(iii) Identifiers must be combination of letters and digits but it must starts with letters.

For ex : (a) Ankush123 ✓ (valid) (b) 123 Kiran × (Not valid).

## Conventions For Identifiers :-

- conventions are not mandatory but According to the Java standard practice we have to Follow.

conventions to Follow :

(i) While creating the class, 1st letter should be in uppercase.

(ii) While creating variables and methods, 1st letter should be in the lowercase.

NOTE :- If the variable or methods are combination of words then 1st letter of 1st word should be in lower case and after that the 1st letter of 2nd, 3rd... word should be in the upper case.

(iii) While writting the classes, 1st letter should be in upper case and if it is combination of words then 1st letter of each and every word should be in uppercase.

For Ex : Sample , DemoTest , Rita geeta sita ⇒ (uppercase) Rita GeetaSita ✓ ✗

**Summary :-** In examples,

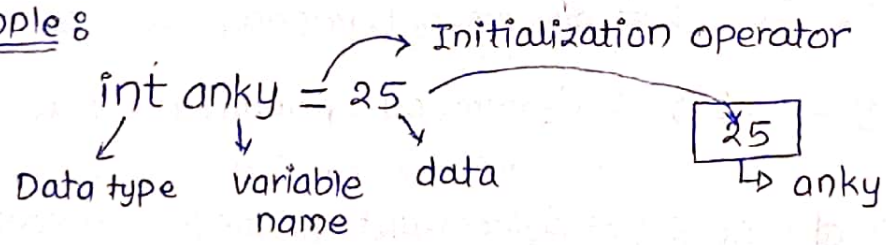
(1) classes :- { Sample , Sample Dimple , SampleTestDemo }

(2) variables :- { rinky , kiranNand , rita Geeta Sita }

(3) Methods :- { sample ( ) , sampleDimple ( ) , sampleTestDemo ( ) }

( ) → parenthesis

Example :




Program :-

```
Package Key_Identifiers
Public class Kiran {// class body open
Public static void main (String[] args){; // main method body open
    int anky = 25 ;
    System.out. println (anky) ; //25
    }// main method body closed
    }// class body close.
```

# Variables :- It is a piece of memory which is used to store a single data. One variable can store only one data at a time.

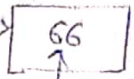
Rules for variables :-

(i) Variable Declaration ⇒ Data type (variable name) ; ← seperator.

(ii) Variable Initilization ⇒ variable name = data ;

For ex : Variable Declaration ⇒ int i ;

Variable initilization ⇒ i = 60 ;

Program : (1)
```
int i ; // variable declaration
i = 66 ; // -"- ---- initialization
```



```
syso (i) ; // 66
```

(2)
```
int i = 66 ; // [variable Declaration and variable initialization within single statement]
syso (i) ; // 66
```

Examples of variables :

(i) byte ⇒ byte e = 10 ; ⇒ System.out.println (e) ; // 10

(ii) short ⇒ short s = 20 ; ⇒ System.out.println (s) ; // 20.

(iii) int ⇒ int i = 30 ; ⇒ System.out.println (i) ; // 30

(iv) long ⇒ long g = 40L ; ⇒ System.out.println (g) ; // 40

(integer; ↳ (L))

(v) double ⇒ double d = 56.5 ; ⇒ System.out.println (d) ; // 56.5

(↳ decimal)

(vi) float ⇒ float t = 50.5f ; ⇒ System.out.println (t) ; // 50.5

(↳ decimal ↳ (f))

(vii) char ⇒ char c = 'A' ⇒ System.out.println (c) ; // A

(viii) Boolean ⇒ Boolean b = true ⇒ System.out.println (b) ; // true.

and

String ⇒ String s = "Kiran" ⇒ System.out.println (s) ; // Kiran

## Types of variables :-

### Local variable

For Ex :

```
class A {
main (-) { //
int a = 20 ; // variable is inside
                the method block
} //
}
```

**Def:** The variable which is declared inside the method block and inside the class block is called as the "local variable".

### Global Variable

For Ex :

```
class A {
int a = 20; → variable is
main (-) { //  outside the
                method
} //            block.
}
```

**Def:** The variable which is declared outside the method block but inside the class block is called as "Global variable"

Program :-

```
Package Variables ;
Public class A { // class body open
Public static void main (string [] args) { // main method body open
/** int i ; // variable declaration
    i = 66 ; // variable initialization
    System. out. println (i) ; */
// variable declaration and initilization in single statement
    int i = 66 ;
    System. out. println (i) ;  // 66
} // main method body close
} // class body close.
```

# classification of variables :-

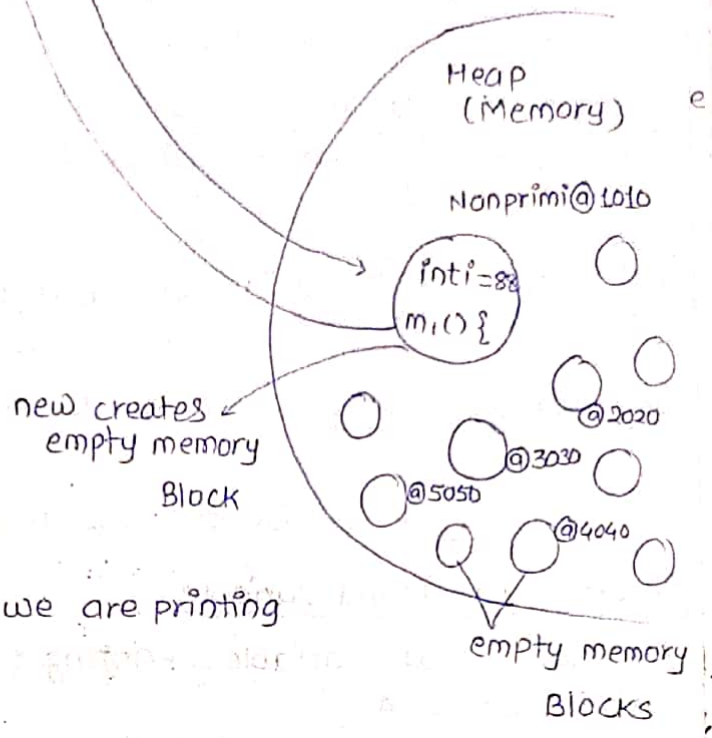
classification of variables

1. Depending on data which is used
   - (i) Primitive variable
     Ex: int i = 10;
     It stores data
   - (ii) Non-primitive variable
     Ex: Sample s = new Sample();
     it stores address
2. Depending upon scope and visibility.
   - Local variable (means declared within method block and class block)
     - scope is always inside the method
   - Global variable (means declared outside the method block & inside the class block)
     - scope is throught the class.

- Static variable / class variable :
  If static keyword is there inside the variable it is called as the static variable.
  For Ex: static int j = 50; (className . variablename)
- Non-static variable / Instance variable :
  If static keyword is not there inside the variable then it is called as Non-static variable. (we have to create object)
  ⇓
  (referencevariable . variablename)
- Non-primitive :
  object : className variablename = new className();
  Nonprimi n = new Nonprimi();
  - Nonprimi → className
  - n → refvariable
  - new → keyword / operator
  - Nonprimi() → constructor.
  - = → Initialization operator

For Example :→ Nonprimi donkey = new Nonprimi () ;

refvariable




```
{
  int i = 88
  Public void m1() {
  }
}
```

Rules For local variable :→

(i) we are printing variables means we are printing data inside it.

Ex: int i = 20 ; → data
(data type, variable)

syso (i) ; // 20.

(ii) Without declaring and initializing variable we can't print variable.

int a ; // no data declared

syso (a) ; // then no output.

(iii) We can reintialized local variable any no. of time but previous data can be deleted and new data will be updated.

Q) Why CTE occurs? CTE - compile time error.

↳ If your program is not as per syntax.
↳ Format of Java program.

for Ex: int c ; // variable declaration. } ← Here will be error as per syntax.
c = 20 ; // variable initialization }

Program:

```
Package variables ;
Public class Local_Rules{;
Public static void main (String[] args) {
// rule 1 we are printing data means we are printing data inside it
//  int a = 55 ;
//  syso (a) ; // Rule 1.
//  rule 2 without initializing we can't print it.
//  int i ;
//  syso (i) ; // No initialization.
// rule 3 reintialization
   int c = 10
     c = 20  ⇒ syso (c) ; //20  }
```

## Local Variable

- The variable which is declared inside the method and inside the class is called as local variable
- There is no concept of static and non-static for local variables.

## Global Variable

- The variable which is declared outside the method but inside the class is called as global variable.
- There is a concept of the static and non-static for global variable.

- For static :- To print use (i) directly with variable name
  (ii) classname.variablename

and For Non-static :- To print use (i) step-I : create the object
(ii) step-II : syso (refvariable.variable name

Object : Sample s = new sample();

## Program for local variable

```
Package Local_variable_printing;
Public class A {
Public static void main (String[]args){
 int i = 10; // local
System.out.println(i); // 10
 int a = 20;
System.out.println(a); //20
}
}
```

## Program for global variable

1.
```
Package Global_Variable_Printing
 public class A {
 public static void main(-){
 static int a = 100; // Global static variable
 public static void main (-) {
 syso (A.a); // classname.varname
 OR
   syso (a); // directly printing.
 }
 }
```

2.
```
Package Global_Variable_Printing
 public class B {
  int a = 200; // Non-static global variable
 public static void main (-) {
```

Classname refvar = new classname(); ⇐ Object ⇐ `B shanky = new B();`

```
 syso (shanky.a);
```

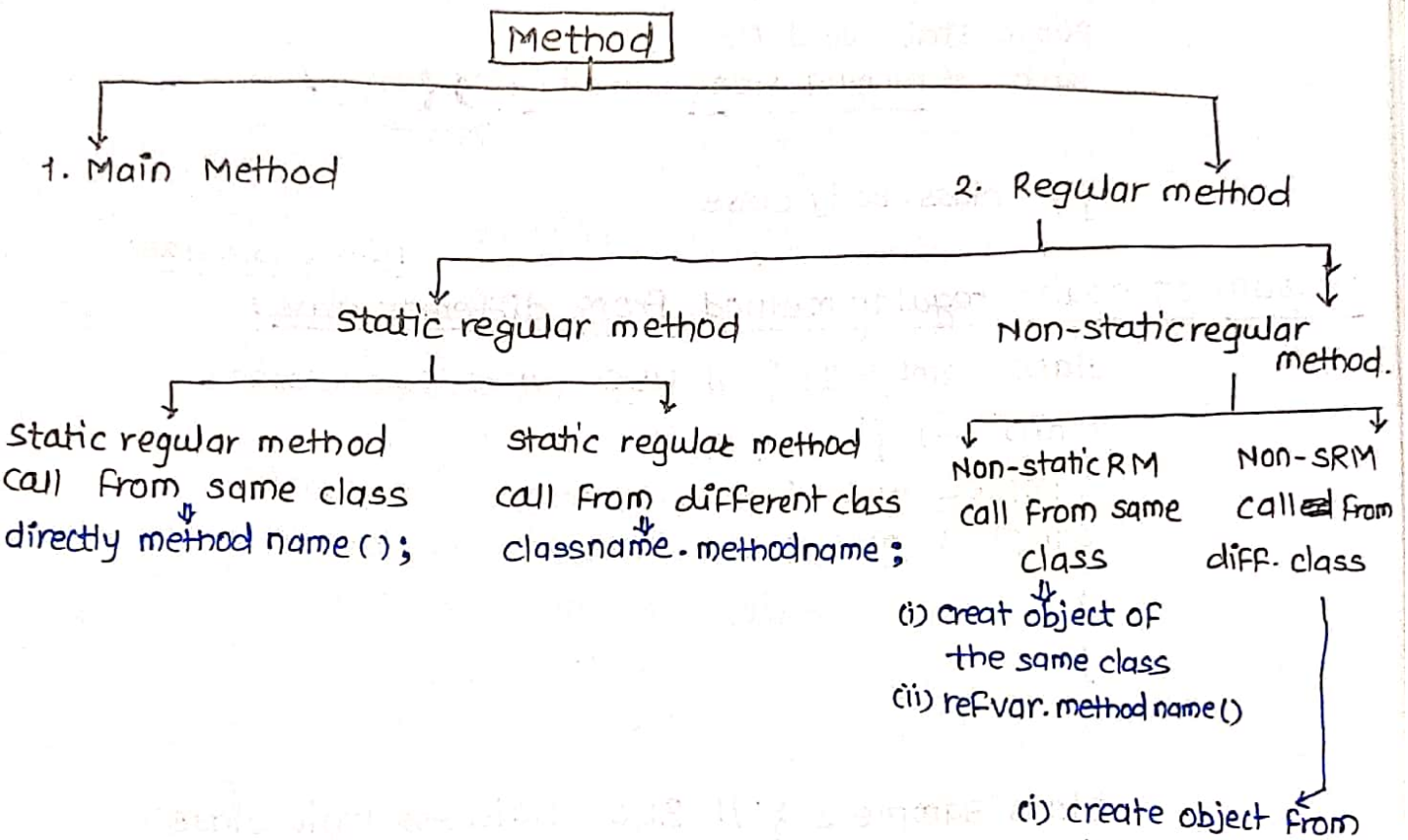
`System.out.println(refvar.variable name);`

```
 }
 }
```

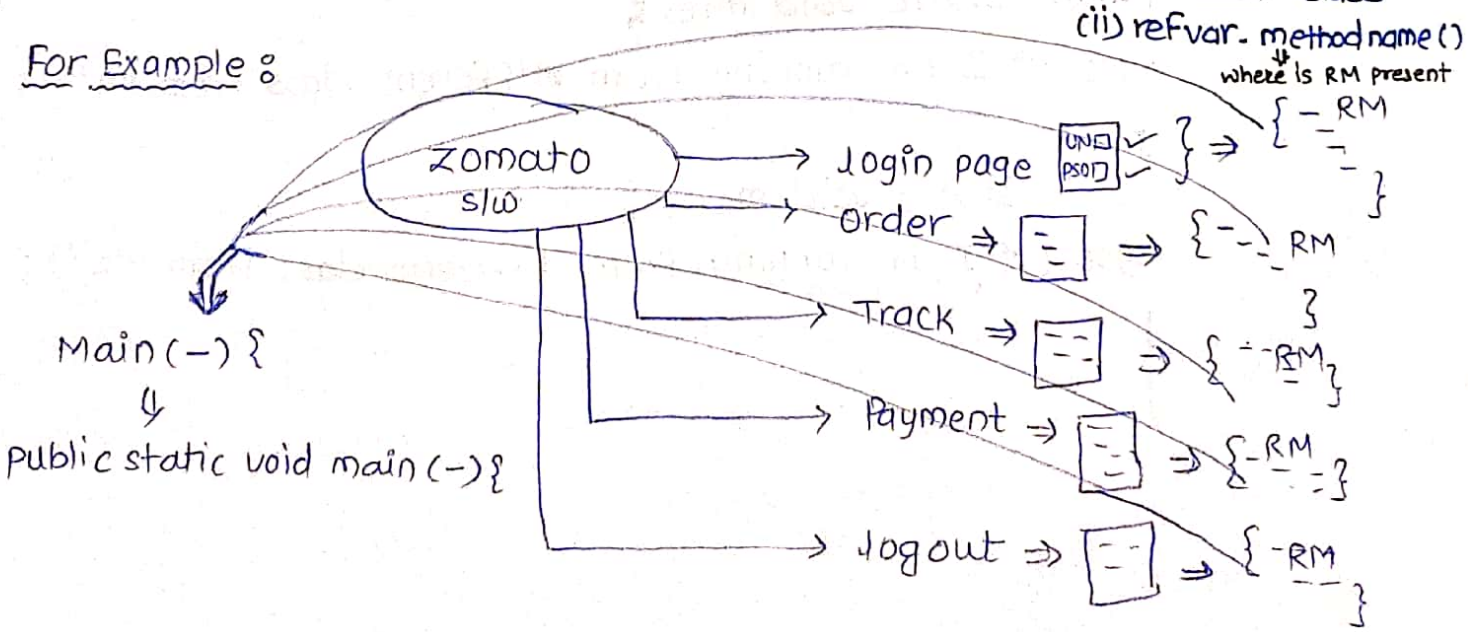
# Methods :- It is a block of code which runs when we call it.

(i) It is used for code reusability purpose.
(ii) It will reduce the length of programms.
(iii) Method is also called as function because each code inside the method performs specific function.

- There are two types of methods.
  1. Main method
  2. Regular method




For Example :-



Program:

```
// static regular method from same class :
class sample 1 { // class body open
main (-) { // main method body open
m1 () ; // method calling statement
m2 () ; // method calling statement
} // main method body close
Public static void m1() {
syso ("running from method m1") ;
}
public static void m2 () {
syso ("running from method m2") ;
}
} // class body close
```

I am calling regular method m1() in main method.

Date: 13/06/2022

Program on static regular method from different class:

```
class sample 2 { // ULC (user logic class)
main (-) {
sample2. m1 () ;
sample 2. m2 ();
}
}
```

```
class sample 2 { // BLC (Buisness logic class)
Public static void m1() {
syso (" I am running from different class from m1");
}
public static void m2 () {
syso (" I am running from different class from m2");
}
}
```

Program on non-static method call from same class:

```
class sample 3 { // ULC // class body open
main (-) {
 sample 3 S = new sample 3 () ; // object
    S. m4 () ; // method calling statement
    S. m3 () ; // method calling statement
}
  public void m3 () {
  syso (" I am running from method M3);
  }
  public void m4 () {
  syso (" I am running from same class with method m4);
  }
  } // class body close
```

Program on Non-static method call from different class:

```
class sample 4 { // ULC (user logic class)
main (-) {
  sample 5 S = new sample 5 () ; // object
   S. m5 () ; // Method calling statement
   S. m6 () ; // method calling statement


}
}
Class sample 5 { // BLC (Buissness logic class)
 public void m5 () {
 syso (" I am running from m5);
 }
 public void m6 () {
 syso (" I am running from m6);
 }
 }
```

## programms with & without patameter :→ (Method)

(i) program without parameter method :

```
class sample { // ULC
main (-) {
addition ( ); // method calling statement
}
public static void addition () {
        int a = 10;
        int b = 20;
        int c = a+b;
        syso (c); //30
}
}
```

(ii) Programs with parameter method :

```
package methods;
// Method with parameter
public class 7 {
main (-) {
addition (20,50); // method calling statement
}
public static void addition (int a, int b) {
        int c = a+b;
        syso (c);
}
}
```

→ compile time binding
Matching formal & actual arguments.

⊞ Compile time Binding :→ The process of matching Formal arguments and actual arguments during compilation is called as "compile time binding".

- It is automatically done by compiler.

Functions of compiler :→

(i) It will converts sourcefile into .classfile

(ii) It will match the formal arguments and actual arguments.

# Constructors :-

14/06/2022

- It is the special type of method which is used to initialize the variables during object creation.

- Rules for constructor :-

(i) constructor name is always same as that of classname. why?
⇒ To recognize the class because in heap area (memory) there are no. of classes so to recognize given/present class we use constructor name as that of classname

(ii) constructor do not have any return type.

(iii) we can write many no. of constructors within single class but their parameters should be different.

- Uses of constructors :-

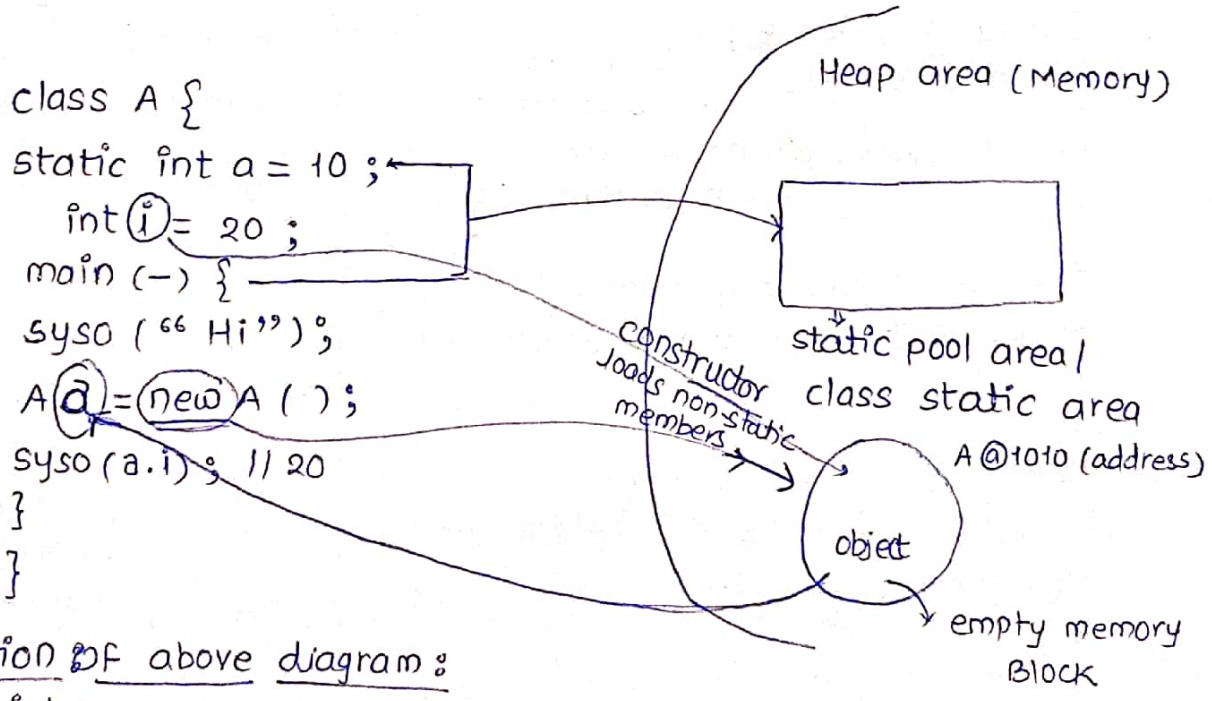
(i) It is used to initialize the variable during object creation.

(ii) It will load non-static members of the class inside the object which is created by new keyword.

Q.] can we make constructor as public ?
↳ Yes ... !

Q]. can we make constructor as private ?
↳ Yes ... - !

```
class A {
static int a = 10 ;
  int i = 20 ;
main (-) {
syso ("Hi") ;
A a = new A ( ) ;
syso (a.i) ; // 20
}
}
```




Explaination of above diagram :

- static int a = 10 ; and main (-) will create the static pool area or class static area.
- In object A a = new A ( ) ; , the new keyword / opetator will create the object in the heap area (Memory). The members in the object will connected to the reference variable i.e. a and the address created by keyword new is attached to the reference variable a.

## # object :-

(i) It is a empty memory block created by new keyword during the object creation statement.

(ii) It will give the address to the object and it is stored in reference variable.

(iii) constructor will load non-static members of the class inside the object.

### Types of Constructor

- Default constructor (It is provided by compiler)
- User defined constructor
  - zero parameter constructor
    Ex: A a = new A ( )
    ↳ No parameter
  - parameterized constructor (there will be a parameter)

### Example of parameterized constructor :

```
public class Gun { // BLC
String gunName ;
int noOfBullets ;   // variable declaration
public Gun (String gunName,
            int noOfBullets) {
this.gunName = gunName ;
this.noOfBullets = noOfBullets ;
}
public shoot ( ) {
For (int i = 1 ; i ≤ noofBullets ; i++){
 Syso ("Deshkew") ;
}
}
}
```

```
class PubG { // ULC
 main (-> {
 Gun g1 = new Gun ("AK47",6);
  g1.shoot ( ) ;
}
}
```

⇒ O/P ⇒ Deshkew 1
Deshkew 2
Deshkew 3
Deshkew 4
Deshkew 5
Deshkew 6

- How program ~~runs~~ runs ?

1,2,3,4,5,6

For (int j = 1 ; j ≤ 6 (no.of Bullets) ; j++ }

↳ I++ is increment operator it will increase value by 1 each time.

j = 1 ; j ≤ 6 ✓
j = 2 ; j ≤ 6 ✓
j = 3 ; j ≤ 6 ✓
j = 4 ; j ≤ 6 ✓
j = 5 ; j ≤ 6 ✓
j = 6 ; j ≤ 6 ✓
j = 7 ; j ≤ 6 X (Not valid)
(7 ≤ 6) X

O/P ⇒ Deshkew 1
—11— 2
—11— 3
—11— 4
—11— 5
—11— 6

15/06/2022

# Conditional Statements :

- Conditional statements are used to perform the operation based upon the output of condition.
- we have four types of conditional statements.

(i) if statement
(ii) if else statement
(iii) else if ladder statement
(iv) switch statement.

(i) if statement :- If block will executes whenever output of the condition is true.

Syntax :- if (condition) {
    =
    =
    = } - statement

For Ex : if (5 > 7) { ↓ False }
    syso (" Hello ");
    X (will not print as condition is false)
    }

and

if (7 > 5) → condition which is true
    syso (" Hello GM ");
    } // Hello GM will be print as Condition is true.

(ii) if else statement :-

- Whenever we have to perform two sets of operation then we have to use if else statement.

Syntax :

```
if (condition) {
 statements ;
}
else {
statements ;
}
```

Example :-

```
class C {
    main (-) {
    int age =    ;
if ( age >= 18) {
syso ("Eligible For License") ;
}
else {
syso ("Not Eligible For License");
}
}
}
```

(iii) if else ladder statements :-

- If we want to perform more than two sets of operations then we have to use if else ~~ladder~~ ladder statement.

Syntax :-

```
if (condition) {
 statements;
 }
elseif( condition) {
 statement ;
 }
else if (condition) {
  statement ;
 }
else if (condition) {
  statement ;
}
else {
statements ;
 }
```

Example :

```
    class Exam {
     main (-) {
    int marks =     ;
if ( marks >= 65) {
syso ("1st class with distinction");
}
else if ( marks >= 60) {
syso ("1st class pass");
}
else if ( marks >= 50) {
syso ("2nd class");
}
else if ( marks >= 35) {
syso ("Pass");
}
else {
syso ("Fail");
}
}
}
```

(iv) Switch :→ 16/06/2022

- switch performs equality checking between the output of the expression and case value.

Syntax : switch (" ") {
↳ expression (output of expression)

```
case " " : {
syso (" ");
break ;
}
case " " : {
syso (" ");
break ;
}
default : {
syso (" ");
}
```

Example : Hostel

```
Class Hostel {
main (-) {
switch ("Idle") {
        ↳ o/p of expression
case "Dosa" : {
syso (" on Monday") ;
break ;
}
case "Wada" : {
syso (" on Tuesday");
break ;
}
case "Idle" : {
syso (" On Wednesday");
break ;
}
default : {
syso (" NO Breakfast");
}
```

- NOTE :→ switch (" ") {
↓
⊛ Expression will allow only
(i) Int
(ii) string
(iii) char
⊛ It will not allow
(i) float
(ii) double
(iii) boolean

# LOOPS :-> Whenever we want to perform same operation for number of times without repeating the statements again and again so we go through the concept that is called as "loops".

- There are four types of loops.
  (i) while
  (ii) do while
  (iii) For loop
  (iv) for each loop.

  } They all are keywords so they starts with lowercase.

(i) while :-> Here the statements of the loop gets executed for many no. of times until the condition is false.

Example :->

```
Class A {
main (-) {
int i = 10 ;
while ( i ≤ 20) {
syso ( i++) ; // 11
}
}
}
```

condition → JVM

10 ≤ 20 → T

Working :

i ≤ 20
⇓
10 ≤ 20 ⇒ T ✓ // 11

i++ ⇒ 11 ≤ 20 ⇒ T ✓ // 12
(increment)

-11 ⇒ 12 ≤ 20 ⇒ T ✓ // 13

-11 ⇒ 20 ≤ 20 ⇒ T ✓ // 20

-11 ⇒ 21 ≤ 20 ⇒ F ✗ // false.

i++ ⇒ increment operator
int i = 10 ← (It will increase value by 1)
syso (i++) ; // 11

i -- ⇒ Decrement operator
int i = 10 ↓ (decrease value by 1)
syso (i--) ; // 9

17/06/22

(ii) do while :- In case of do while loop, we are writting executable statements of do block and it will execute on the basis of condition inside while block.

For Example :

```
Class do_while_Example {
  main (-) {
  int i = 10 ;
 do {
 syso (i++); //11
 }
 while (i<=20) ;
 }
 }
```

JVM

↳ seperator

21 <= 20 ⇒ False (NO printing)

- Note :- If we are writting statements after while seperator then that will executes as usual.

Example 2 :

```
class {
main (-) {
int a = 10 ;
do {
 syso (a) ; //10
 a++ ;
}
while (a <= 20) ;
}
}
```

① JVM

② after it will increment

then JVM move to while

Execution process

a=10

a<=20

10 <= 20 ⇒ T ✓

a++

11 <= 20 ⇒ T ✓

12 <= 20 ⇒ T ✓

13 <= 20 ⇒ T ✓

!

!

20 <= 20 ⇒ T ✓

21 <= 20 ⇒ F X.

It will stop execution.

(iii) For loop :▷ (VIMP) ★★★

Here we are writing variable declaration / variable initialization and condition as well as updation within single statement.

Syntax :- for ( variable declaration / initialization ; condition ; updation)

Example : for ( int i = 1 ; i ≤ 5 ; i++ ) {

- int i = 1 ⇓ : Variable declaration and variable initialization
- i ≤ 5 → condition
- i++ → Increment operator. (updation)

---

Q.] what is the difference between sop() and sopln() ? (printing statements).

↳

| sopln() [system.out.println()] | sop() [system.out.print()] |
|---|---|
| (i) 1st it will print the data and cursor moves to the next line. | (i) 1st it will print the data and cursor move next in the same line. |
| (ii) Here data is not mandatory inside the printing statements. | (ii) Here data is compulsory in the printing statements otherwise it will give the compile time error (CTE). |

---

Examples of For loop :-

Q.] Print the 5 ★ within single line ?

↳

```
class A {
main (-) {
for ( int i = 1 ; i ≤ 5 ; i++ ) {
System.out.print ( " ★ " ) // ★★★★★ (o/p)
}
}
}
```

Execution process :

i = 1

i ≤ 5

1 ≤ 5 ⇒ T ✓ ★ ★ ★ ★ ★ = 5★

2 ≤ 5 ⇒ T ✓

3 ≤ 5 ⇒ T ✓

⋮

6 ≤ 5 ⇒ F ✗

Q] Print 10 stars on different lines?

⇒
```
for (int i=1 ; i≤10 ; i++) {
system.out.println("*") ;   // *
}                              *
                               *
```

Execution process

i = 1

i ≤ 10

1 ≤ 10 ✓

2 ≤ 10 ✓

⋮

11 ≤ 10 ✗ stops execution

# Nested for loop :-

- Writing one for loop inside another for loop is called as Nested for loop.

Example:

```
for (int i = 1 ; i≤5 ; i++) { // ① written for rows
    for (int j = 1 ; j≤4 ; j++) { // ② written for coloumns
```

for → outer loop

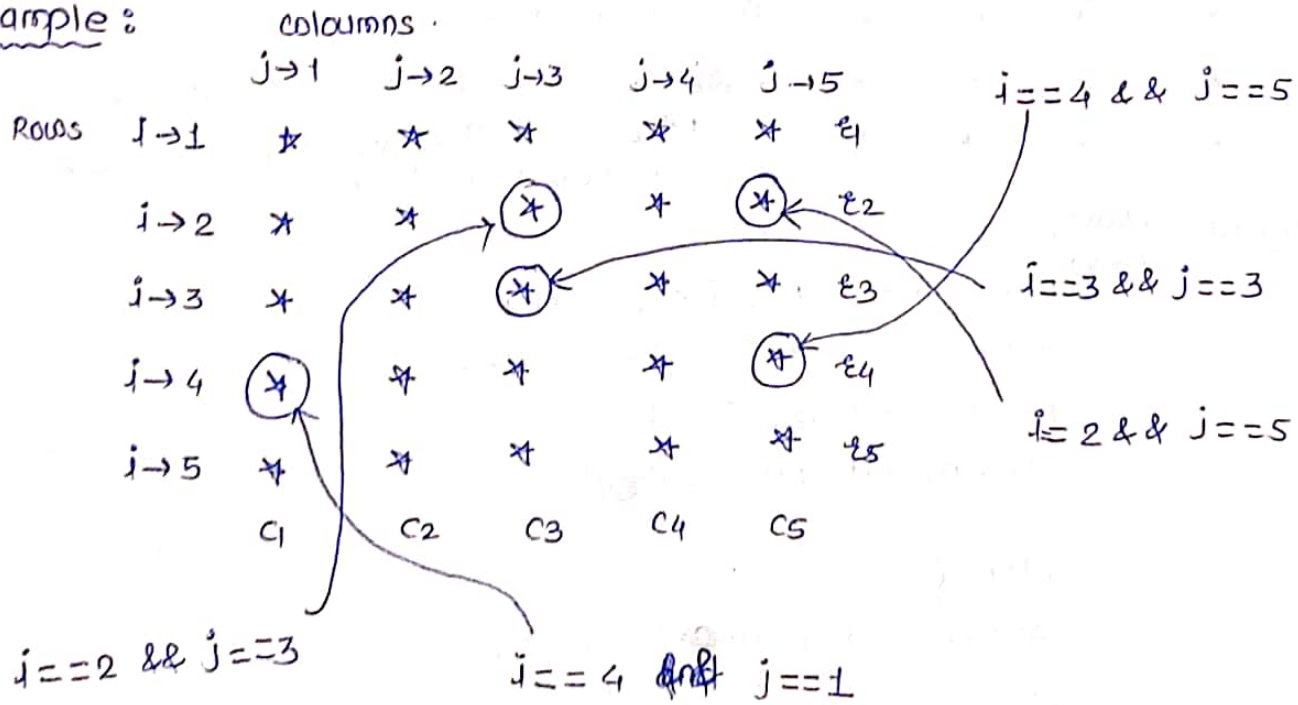
for → inner for loop

Execution :- For each and every iteration of outer for loop, inner for loop executes completely.

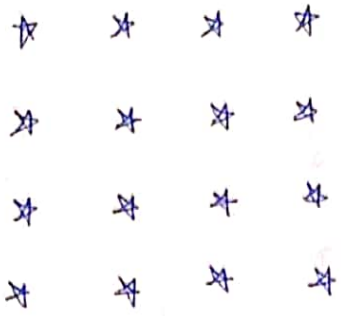
- syntax :

```
for (int i = 1 ; i≤5 ; i++) { ⇒ 5 rows
for (int j = 1 ; j≤5 ; j++) { ⇒ 5 coloumns
```

Example :



E.g. (1) ⇒ no. of rows = 4
no. of columns = 4



Programm :

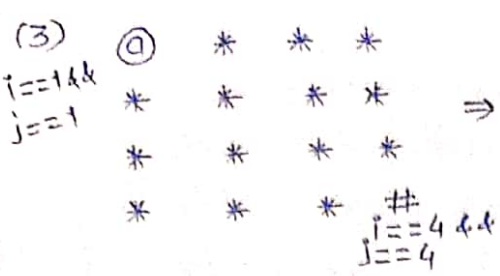
```
class A {
main (-) {
For ( int i = 1 ; i ≤ 4 ; i++) { // outer For loop for row
For ( int j = 1 ; j ≤ 4 ; j++) { // inner for loop for colum
system. out. print ("*") ;
}
system. out. println ( ) ;
}
}
}
```

(2) ⓐ * * * no. of rows = 4
i=1, j=1
* * * * no. of columns = 4
* * * * - we have to do two operations
* * * * 1st operation ⇒ @ } use if else.
2nd operation ⇒ *

Program :

```
class A{
main-) {
For (int i=1 ; i ≤ 4 ; i++) {
For ( int j=1 ; j ≤ 4 ; j++) {
if ( i==1 && j==1){
system. out . print ("@") ;
}
else {
system . out . print ("*") ;
}
system. out. println ( ) ;
}
}
}
```

(3)



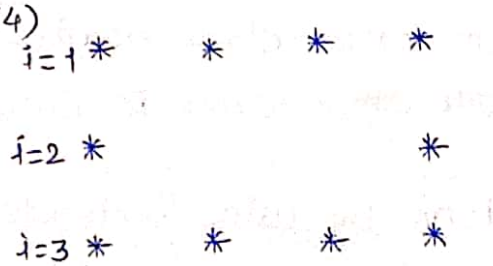

No. of rows = 4 , No. of columns = 4

We need 3 sets of operations.
(if else ladder)

Program:

```
Class A {
main (-) {
For (int i=1 ; i≤4 ; j++) { // outer For loop for rows
For (int j=1 ; j≤4 ; j++) { // inner // ——— columns
if (i==1 && j==1) {
system.out.print ("@") ;
}
else if (i==4 && j==4) {
system.out.print ("#") ;
}
else {
system.out.print ("*") ;
} // closed else block
} // inner For loop.
system.out.println () ;
}
}
}
```

(4)

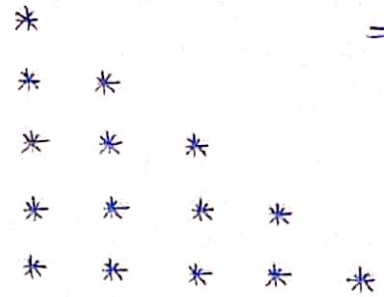



⇒ no. of rows = 3
no. of columns = 4

Program:

```
For (int i=1; i≤3 ; i++) { // outer
For (int j=1 ; j≤4 ;j++) { // inner
if (i==1 || i==3 || j==1 || j==4) {
system.out.print ("*") ;
}
else {
system.out.print (" ") ;
}
system.out.println () ;
}
```

(5)



⇒

Program:

```
class A {
 main (-) {
int star = 1 ;
For (int i = 1 ; i<=5; i++) {
For (int j = 1 ; j<=star; j++) {
   sop ("*") ;
}
   sop ( ) ;
   star ++ ;
}
}
}
```

# OOPS :→

- These are the principles which are used to convert the real time sceneries (object) into Java programming.
- There are four types of oops principle.

1. Inheritance
2. Polymorphism
3. Encapsulation
4. Abstraction

} These Four oops principles are called as Pillars of Java.

1. Inheritance : The process in which one class aquires or gain properties from other class is called as Inheritance.
   - Inheritance can be done by using "extends" keyword.

- Superclass / parent class :
  The class from where we are aquiring the properties is called as superclass.
- Subclass / child class :
  The class in which we are aquiring the properties is called as subclass.
- Types of Inheritances :
  (i) single level (ii) Multi-level inheritance
  (iii) Multiple inheritance (iv) hirarchicle inheritance.

Program of single level inheritance :

⇒
```
Package inheritances;
Public class Father { // BLC
public void money () {
System. out .println ("50 Lakhs");
}
public void car () {
syso ("SKODA");
}
public void home () {
syso ("1 BHK");
}
}
}
```

⇒
```
Package inheritances ;
Public class son { // BLC
↳ extends Father
public void Bike (){
syso ("Bullet");
}
}
```

→ To copy or to take properties from father class extend is used.

⇒
```
Package inheritances ;
Public static void main ( String [] args ) { // ULC
syso ("All the properties of son are");
son S = new son ();
S. money ();
S. car ();
S. home ();
S. Bike ();
}
}
```

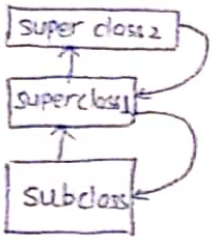
⇒ O/p → All properties of son are
50 Lakhs
SKODA
1 BHK
Bullet

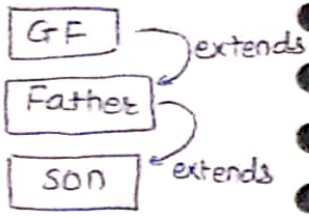
NOTE :→ For single valued inheritance 2 classes are mandatory.

(ii) Multi level Inheritance :- (3 classes are mandatory)

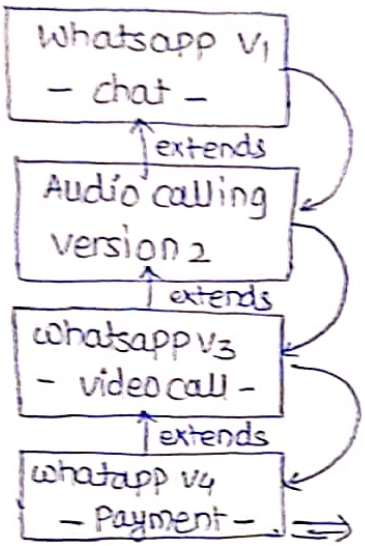
one subclass aquires the properties from another superclass and this superclass aquires properties from a particular superclass.

→ subclass = super class1 + super class2




Example :- Whatsapp




⟹ Chat + Audiocall + Videocall + Payment

Program : whatsapp.

Class1 ⇒ (BLC)

```
Package Inheritance ;  // class 1
Public class whatsapp_V1 {
Public void chat() {
System.out.println(" Hi, Hello, Chatting");
}
}
```

Class 2 ⇒ (BLC)

```
Package inheritance ;
Public class whatsapp_V2 extends whatsapp_V1 {
Public void audiocall ( ) {
Syso (" Talking gulu gulu");
}
}
```

Class 3 ⇒ (BLC)

```
Package inheritance ;
Public class whatsapp_V3 extends whatsapp_V2 {
Public void videocall ( ) {
syso (" video.talking");
}
}
```

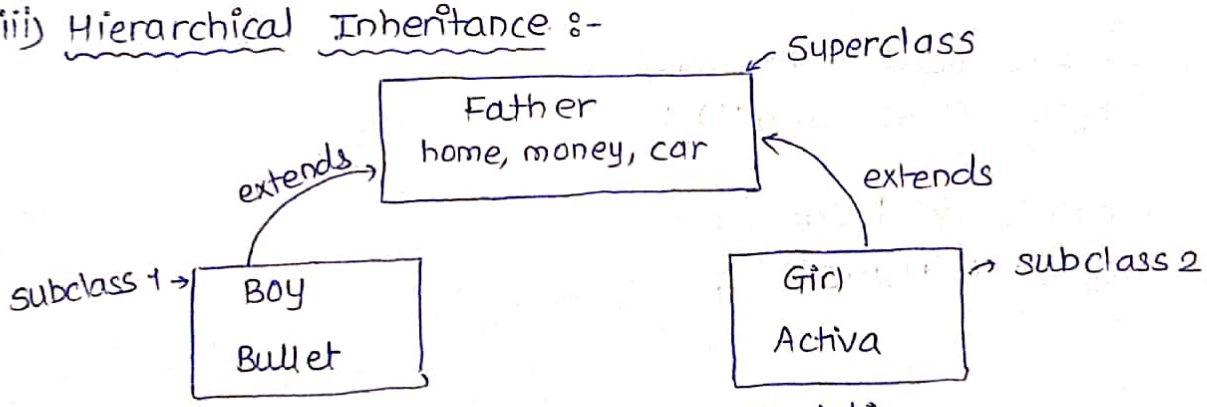
Class 4 ⇒ (BLC)

```
package inheritances ;
public class whatsapp_V4 extends whatsapp_V3 {
public void Payment () {
syso (" Bol Bhai kiti rupaye pahijet ");
}
}
```

Main class ⇒ (ULC)

```
public class multilevel_Inheritance {
main (-) {
syso (" All versions of whatsapp ");
whatsapp_V4 V5 = new whatsapp_V4 ();
V5. chat ();
V5. audiocall ();
V5. videocall ();
V5. payment ();
}
}
```

(iii) Hierarchical Inheritance :-




1. Bullet
2. home
3. money
4. car

1. Activa
2. home
3. money
4. car

- Hrerarchical inheritance is a type of inheritance in which multiple subclass aquires the properties from single super class.

Program :-

```
Package Hierarchical_Inheritance ;  // class 1 // BLC
Public class Boy extends Father {
Public void Bullet () {
Syso ("Hello") ;
}
}
```

Class 2 :→

```
Package Hierarchical_Inheritance ; // BLC
Public class Girl extends Father {
Public void Activa () {
Syso ("Hi") ;
}
}
```

Class 3 :-

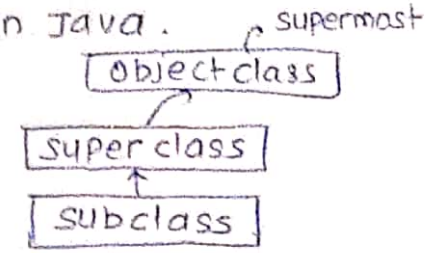
```
Package Hierarchical_Inheritance ; // BLC
Public class Father {
Public void home () {
Syso ("1 BHK") ;
}
Public void money () {
Syso ("50 Lakh") ;
}
Public void car () {
Syso ("Nano") ;
}
}
```

class 4 :-

```
Package Hierarchical_Inheritance ; // ULC
Public class ULC {
main (-) {
Syso ("Properties of Boy are") ;
Boy b = new Boy () ;
b. Bullet () ;
b. home () ;
b. money () ;
b. car () ;
Syso ("Properties of Girl are") ;
Girl g = new Girl () ;
g. Activa () ;
g. home () ;
g. money () ;
g. car () ;
}
}
```

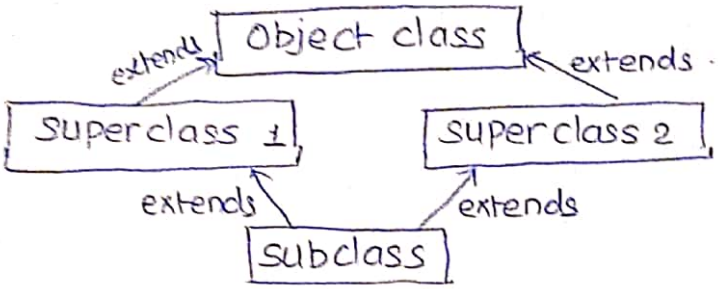
# Object :-

- Each and every class in Java automatically extends to the Object class.
- Object class is the supermost class in Java.
- Object has 11 non-static methods.



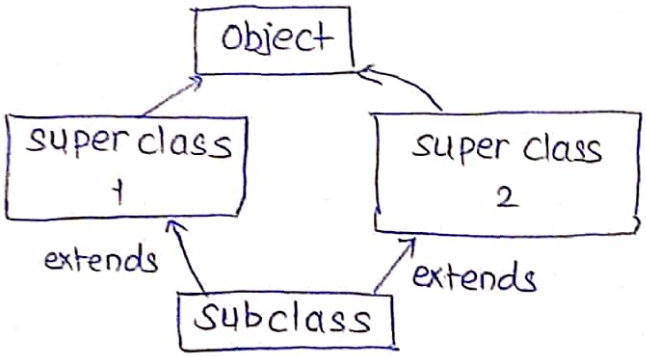

(iv) Multiple Inheritance :

- In this one subclass trying to aquire properties from two superclass.
- It is also known as "Diamond Ambuguity problem" (DAP)




- To achieve multiple Inheritance in Java, we have to take the help of Interfaces.

Question : How diamond ambuguity problem (DAP) occurs ?




- In this, subclass tries to aquire propeties of superclasses. It is not possible to aquire the properties from both the classes at the same time. So subclass will have confusion on that time DAP occurs.
- The process in which the super class statement of subclass constructor is trying to call constructors of two superclass at a time so confusion takes place and DAP occurs.
- [ Actually classes are not calling constructors are calling ].

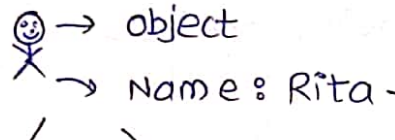
## Polymorphism :-

- It is latin word whose meaning is poly means many and morphism means forms.

poly + morphism = polymorphism
⇓ ⇓
many + forms = Many forms (different)

- The process in which one object showing different behaviour at different stages of life cycle is called as polymorphism.

Example :



→ object
→ Name : Rita

- Market → customer
- college → student
- Ground → player
- cultural activities → performer.

Properties
1) color
2) height
3) weight
4) name

Behaviour
1) studying
2) working

- There are two types of polymorphism.
  (i) compile time polymorphism ⇒ Ex: Method overloading
  (ii) Runtime polymorphism ⇒ Ex: Method overriding.

| Compile time polymorphism | Runtime polymorphism. |
|---|---|
| (i) The binding can be takes place during compilation process | (i) The binding can be takes place during runtime process. |
| (ii) It is also called as early binding. | (ii) It is also called as late binding. |
| (iii) Method overloading is an example of compile time poly. | (iii) Method overriding is an example of runtime polymorphism. |
| (iv) once binding is done then rebinding is not possible. Hence it is called as static binding. | (iv) once the binding is done rebinding is possible. Hence it is called as Dynamic binding. |

Comparison between Method overloading and method overriding.

| Method overloading | Method overriding |
|---|---|
| (i) It is having same method name with different parameters is called as method overloading. | (i) In this method declaration is same but method implementation is different. |
| (ii) Inheritance is not mandatory | (ii) Inheritance is Mandatory |
| (iii) Object creation is not neccessary | (iii) object creation is neccessary. |
| (iv) All static methods overloads | (iv) All non-static methods overrides. |
| (v) It is an example of compile-time polymorphism. | (v) It is an example of the Runtime polymorphism. |

Program :⇒

```
Class Method_overloading {
    main (-) {
addition ( 10, 20); // method calling statement
addition (10, 20, 30); // ==
}
Public static void addition (int a , int b) {
syso ("addition of two no.s :" + (a+b)) ;
}
Public static void addition (int a, int b, int c) {
syso ("addition of three no.s :" + (a+b+c)) ;
}
}
```

(actual arguments → addition(10, 20); addition(10, 20, 30);)
(formal arguments → int a, int b / int a, int b, int c)

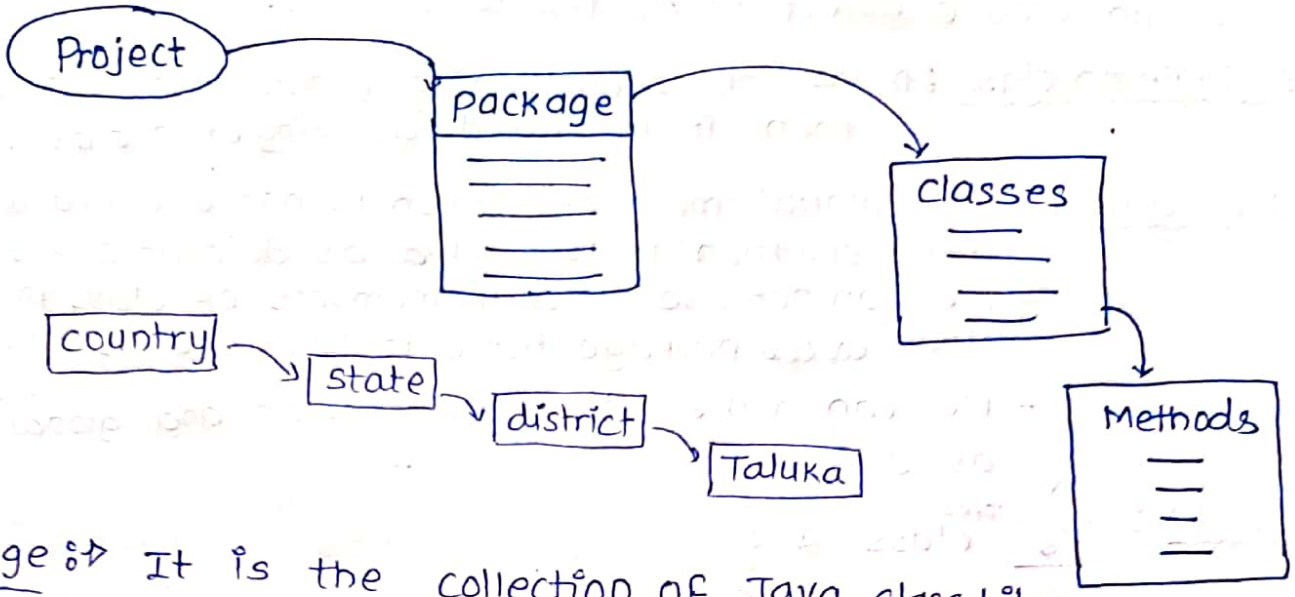
o/p :⇒ addition of two no.s : 30
all ——— three no.s : 60

# Compiletime binding :⇒

The process of binding actual arguments / parameters of method calling statement with formal arguments / parameters of called method is called as CTB.

concatination : [ "addition of no.s :" + ( 30 ) ]

We know that,




Package :⇒ It is the collection of Java class files.

- some important packages in Java :
  (i) java.lang.package ⇒ Object → class
  (ii) java.util.Package ⇒ scanner class
  (iii) java.io. Package ⇒ string
- The fully qualified name of any class is written as,
  syntax : Packagename.classname
  Ex :- java.lang.object
  java.util.scanner
  java.io.string
  } These are Fully qualified names.

## ⊛ Access Specifiers :⇒

- These are used in Java to declare the scope of global members in the class or packages / project.
- There are Four types of access specifiers.
  (i) Private → within class only
  (ii) default → within package (known as package scope)
  (iii) protected → Access from Package to package
  (iv) Public → anywhere we can access.

(i) Private :⇒ If we declare any member of class as private then scope of that members is always inside the program / class.

For Ex:

```
Class A {
    private int a = 10;
    main (-) {
    syso ("Hi");
    }
    }
```

→ has scope within the class only

- we can make constructor as private.
- we can create object inside the same class.

⊛ singleton class :⇒ A class containing constructor as private, then it is called as singleton class.

(ii) default :- Any global members which is not declared with any specifier then it is called as default access specifier.
- we can not use default members of class in any other ~~class~~ package, hence it is called as "package scope".
- we can make class, constructor and global members as default.

For EX : (NO specifier ↓)

```
class A {
main (-) {
syso ("Hi");
}
}
```

(iii) Protected :⇒ IF protected keyword is declared with any global members of class then it is called as protected.
- we can access protected members inside ~~of~~ other package From ~~inside~~ any other package.

Q. Rules to access members from one package to another package?
⇒ (1) we need to import the class containing protected keyword.
(2) we have to create relationship between the classes.
(3) By creating subclass object we can inherit to another class.

# NOTE : (1) we cannot make class as protected.
(2) we can make constructor as protected.

(iv) public :⇒ A global members declared with public keyword are accessible in any other classes, packages, it means it's scope is throughout the project.

Ex : constructors, classes.

Private < default < Protected < Public

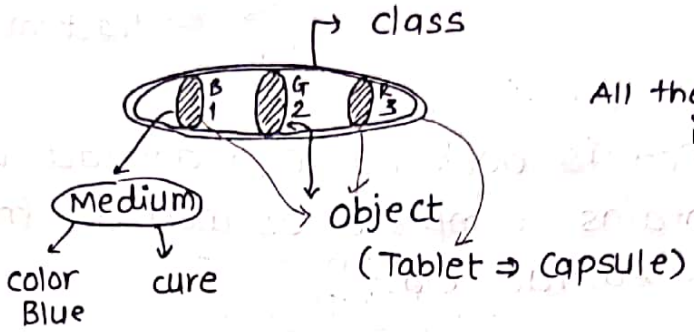
## 3. Encapsulation :▷

The process of wrapping the properties and behaviour of object inside the class is called as "Encapsulation".

Advantages :

(i) code reusability

(ii) we can access private members of the class using setters() and getters().




(objects)
All the mediums are bounded in class (in capsule)

setters() and getters() :

setters() :▷ (i) These are used to reinitialize the private member of Encapsulated class (writting process).

(ii) setters() does not have returntype.

getters() :▷ (i) These are used to fetch the data from the private members of Encapsulated class.

(ii) getters() have returntype depending upon data type

## 4. Abstraction :▷

- The process of hiding the internal implementation code and providing only functionality is called as "Abstraction".
- Abstraction means (incomplete).

[Q] can we achieve 100% abstraction in Java?

⇒ yes....! It is possible using concept of "Interface".

Methods in abstraction :

(1) Complete method :▷ A method contains method declaration and method initialization.

Ex :

```
class A {
main(-) {
Public static void m1() { // Declaration
Syso("Hi"); // Initialization
}
}
}
```

(2) Incomplete method :▷ (Abstract method)

- A method which contains only declaration is called as incomplete method.

For Ex:

```
class B {
main (-) {
public static void m2() ;
```

→ seperator used for incomplete

→ Abstract method / incomplete method

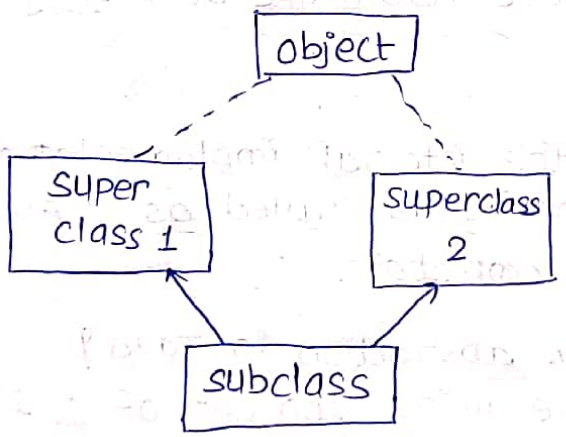
(#) Abstract class :-

A class which is declared with abstract keyword and a class which contains complete as well as incomplete methods is called as "Abstract class".

# Concrete class :▷ It is the class which provides implementation to abstract methods which are present in abstract class. we can implements abstract methods in abstract class by using "extends" keyword.

Q.] How DAP occurs ?

↳▷

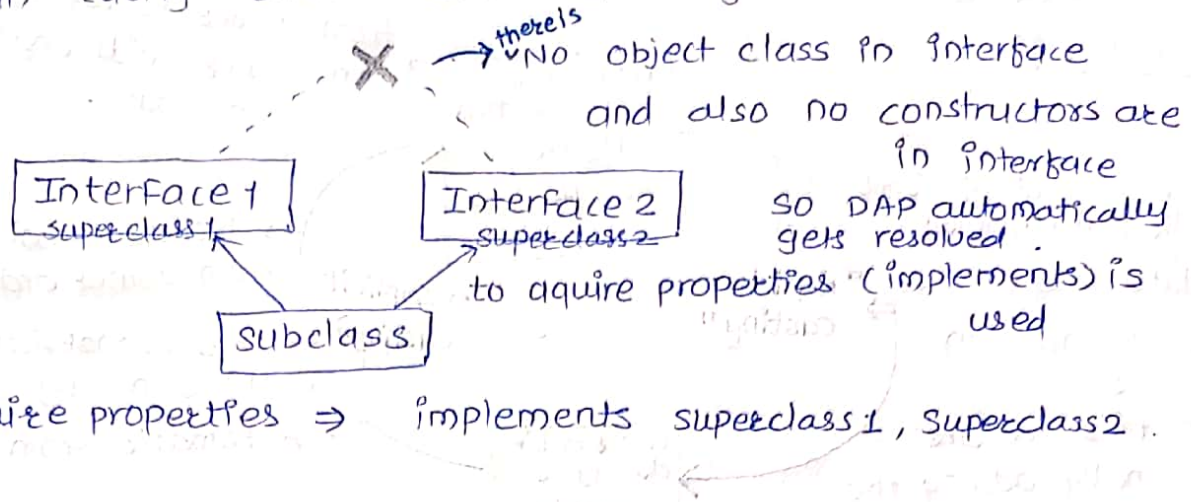



- The subclass's constructor trying to aquire the propeties from two superclasses at the same time. The subclass constructor will have some confusion from which class we have to aquire the propeties. so confusion takes place and then DAP occurs. ( Diamond Ambuguity problem).

- For DAP occurance object class and constructors are responsible.

# How to resolve Diamond Ambugwity problem ? (DAP)

⇒ We can easily remove DAP by using Interfaces.




TO aquire properties ⇒ implements superclass1, Superclass2.

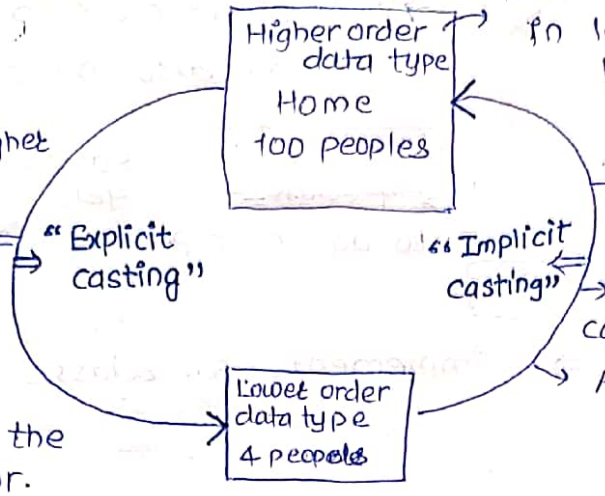
# Interface :-

- It is a oops principle. It is purely abstract in nature.
  (It contains abstract (incomplete) methods)
- we can achieve 100% abstraction by using Interface.
- Features OF Interface :

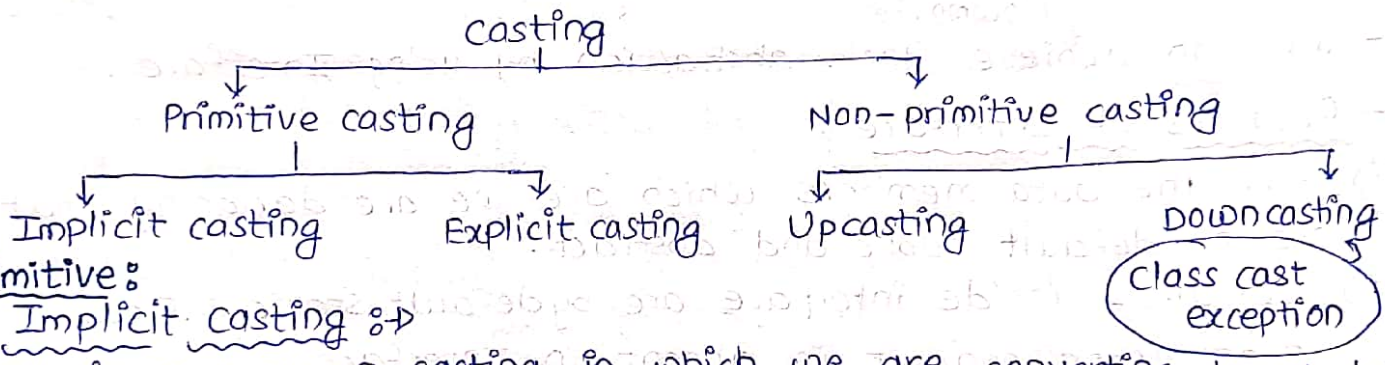
(i) All the data members which are we are declaring that are by default public and abstract.
(ii) variables inside interface are by default static & Final.
(iii) Constructor concept is absent in Interface.
(iv) object of interface can not be created.
(v) Interface supports multiple Inheritance.
(vi) Interface resolves DAP.
(vii) TO create object OF interface we have to use "Implementation class".

# Casting :-

Example :-




- Casting : The process in which one type of information is converted into another type of information or data is called as casting.




Primitive :

(1) Implicit casting :-

- It is a type of casting in which we are converting lower data type into higher data type.
- It is automatically done by compiler from JDK 1.5 onwards.

For Ex : `double d = 10;` — 10 : integer type data (4 byte); d : decimel (8-byte). This integer data is automatically converted into decimel by compiler.

- As the data is converted automatically. Hence it is also known as "Autowidening".

(2) Explicit Casting :-

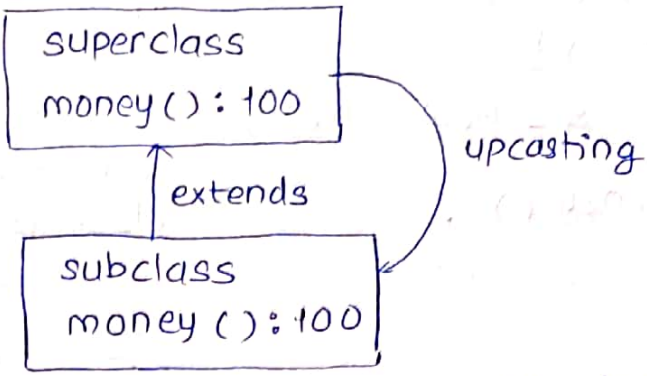
- It is a type of casting in which a higher order data is converted into lower order data type.
- It is done by externally by adding typecast operator.
- As the conversion is done explicitelly, hence it is called as "Explicit Narrowing".

For Ex : `int i = 10.0;` ⇒ `int i = (int) 10.0;`

i : integer type data; 10.0 : double data; (int) : Typecast operator.

## Non-primitive casting :⇒ ( screeshot / scroll handling)

(1) Upcasting :⇒




mobile() → It's newly declared property in subclass.

Upcasting : Assigning subclass property to superclass for updation is called as "upcasting"?

- Before performing upcasting first we need to perform inheritance, after performing inheritance, properties will come from superclass to subclass.
- In subclass we can declare new property during upcasting. so we can print only those properties which came from superclass to subclass.

(2) Downcasting :⇒ Assiging superclass properties into subclass is called as Downcasting.

- Before performing downcasting, upcasting is mandatory.

Note : If we do direct downcasting without performing the upcasting we will get "classcast exception"?

Programs :⇒

```
Package casting ; // class 1
Public class Father { //BLC
public void money () {
syso (" Money : 100") ;
}
}
```

```
package casting ; // class 2
public class son extends Father {//BLC
public void money () { // already available property in superclass
syso (" money : 150);
}
public void mobile () { // newly added property
syso (" I-phone");
}
}
```

```
Package casting ; // class 3
    public class upcasting { // ULC
    main(-) {
    Father s = new son() ; // creation of object of subclass
                              but providing reference of super
     s. money() ;             class is called upcasting.
    }
    }
```

# this keyword and super keyword :→

(i) this keyword : This keyword is always present inside the Non-static context such as non-static method.

- purpose : It is used to access global variables from same class.

Example :

```
class A { // BLC
int a = 50 ; // Non-static global variable
public void test () {
int a = 60 ;
syso (a) ; // 60
syso ( this. a) ; // 50.
```

---

```
class this keyword { // ULC
 main (-) {
 A b = new A () ;
  b. test () ;
 }
 }
```

(ii) super keyword :→ It is used to access global variables from the superclass.

For Ex :

```
class B { // class 1 // BLC
int a = 50; // Non-static global variable.
}
```

```
public class C extends B {//BLC
    int a = 60; // global in same class
public void test () {
    syso (this.a); //60
    syso (super.a); // 50
```

---

```
public class super_keyword { //ULC
public static void main (-) {
    C d = new C ();
    d.test ();
}
}
```

30/06/22

# Main (-) :→ First of all JVM search for SIB, if SIB is not available in program priority goes to main method. JVM starts execution from main (-).

Static initialization Block.

- main (-) is responsible to run other all regular methods.
- If JVM is unable to find main (-) then it results in "No such method Error. main"

Syntax : public static void main (string [] args)

- public: To call JVM from anywhere the public keyword is used.
- static: without creating object JVM can easily access. Hence it is static
- void: It is returntype. it will perform functionality but won't return anything
- main: methodname
- string [] args: data type; command line arguments.

Changes in syntax :→

(i) static public void main (string [] args){
(ii) public static void main (string [] ankush){
(iii) public static void main (string [] args){
(iv) public static void main (string args []){
(v) public static void main (strings..... args){

- Main (-) can overload because we can change the arguments.
- main (-) can not be override ⇒ static have single copy so it will not override.

⊛ SIB (static initialization block) : It is a block which runs before the main (-) method. SIB executes only one time in total process. If SIB is written before or after the main (-), SIB executes firstly. SIB does not have return type.

# Difference between static and non-static :-

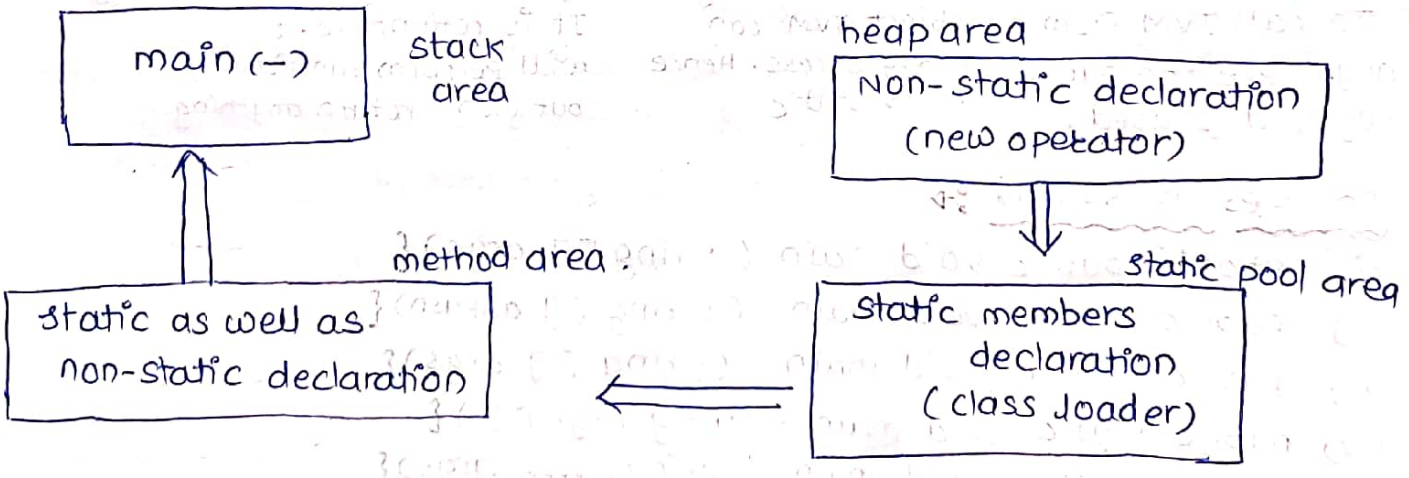
| Static | Non-static |
|---|---|
| (i) They have single copy. | (i) They have multiple copies. |
| (ii) class loader will load all static members in static pool area. | (ii) constructor with new keyword will load all Non-static members into object. |
| (iii) No need of creation of object | (iii) object creation is mandatory. |
| (iv) static members are loaded during loading process of class. | (iv) Non-static members are loaded during object creation. |
| (v) static members overload | (v) Non-static members overide. |

constructor → overload - (we can change parameter hence it can overload)

constructor → override - (i) constructor name is always same as that of classname

(ii) constructor doesn't have any returntype.

## JVM Architecture



01/07/22

# String :- It is a non-primitive data type whose memory size not fixed.

- String is a inbuilt class present inside Java.lang.package.
- String is collection of characters.
- Ex: String s1 = "MAHABHARAT";

(1) final class :- we can not inherit it.

(2) final method :- we can not override it.

(3) final variable :- we can not reintialize.

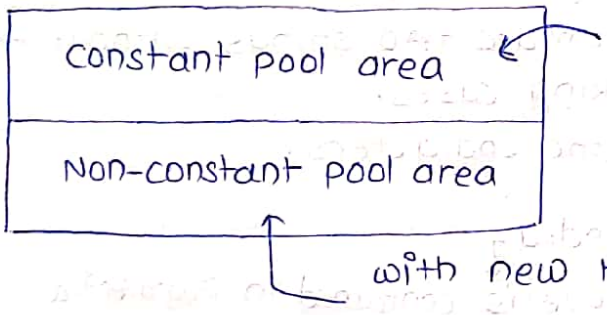
Q] What is the difference between final, finally & finalize()?

A (i) final - it is the keyword

(ii) finalize() - It is non-static method present in object class in which there is Garbage collector.

(iii) finally - It is the block which is present after try and catch block in exception handling mechanism.

(*)




without new keyword object are stored.

with new keyword objects are stored.

For example : String s1 = "Ankush"; ⇒ without new keyword it will stored in constant pool area.

String s2 = new String(); ⇒ with new keyword stored Non-constant pool area.

- String is the final class so that we can't inherit it.
- String objects we can't change (immutable).
- At the time of string declaration, object creation takes place.
- Object creation of string can be done in 2-ways.

(i) without using new keyword (constant pool area)

(ii) with new keyword (Non-constant pool area)

## String Functions / methods :

String s1 = "velocity";
String s2 = "VELOCITY";
String s3 = "velo"

Q. How many characters are there in string s1, s2, s3 or what is the length of the strings?

⇒ syso (s1.length()); // 8
syso (s2.length()); // 8
syso (s3.length()); // 4

```
package string class;
public class A {
main (-) {
string s1 = "velocity";
string s2 = "VELOCITY";
String s3 = "velo";
sysopln(s1.length());
                    // 8
sysopln(s2.length());
                    // 8
sysopl (s3.length());
                    // 4
}
}
```

04/06/22

### String class methods :⇒

- It have some non-static methods.

(1) length() :⇒ It will give / count no. of characters.

(2) toUpperCase() :⇒ It will convert lower case to upper case string.
Ex: velocity ⇒ VELOCITY

(3) toLowerCase() :⇒ It will convert upper case string to lower case string.
Ex: VELOCITY ⇒ velocity

(4) equals() :⇒ It will compare two strings.

(5) equalIgnoreCase :⇒ It will compare two strings without the checking cases.

(6) contains() :⇒ It will check for the characters.

Ex: Raghavendra
endra
→ have is contained in Raghavendra

(7) charAt() :⇒ It will give character of perticular position.

(8) indexOf() :⇒ It will give index of perticular character.

Ex: (A)VELOCITY
0 12345678

(9) startsWith() :⇒ It will check our string is start with the perticular character.

(10) EndsWith() :⇒ It will check our string ends with perticular character.

(11) substrings() :⇒ It is used to break the string.

(12) concat() :⇒ It is used to join the string.

(13) replace() :⇒ It will replace collections of characters.

Ex: velocity
veloRural
Replaced character.

05/06/22

# Diff. between string, string Buffer, string Builder :-

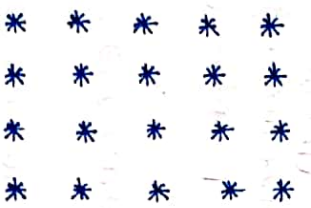
| String | String Buffer | String Builder |
| --- | --- | --- |
| (1) String objects are immutable | (1) String Buffer objects are mutable. | (1) String Builder objects are mutable. |
| (2) objects can be created in two ways. (i) using new keyword (ii) without using new keyword (literal) | (2) object can be created using new keyword | (2) objects can be created using only new keyword. |
| (3) objects can be either stored in heap or string pool area. | (3) objects can be created directly in heap area. | (3) object can be created only within heap area. |
| (4) String is faster. | (4) String buffer is moderate. | (4) string Builder is faster. |
| (5) equals() method compares on the basis of data. | (5) equals() methods compares on the basis of reference variables. | (5) equals() method compares on the basis of reference variable. |

# Star Patterns :

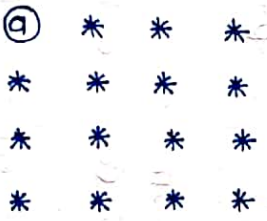
(1)

(2)

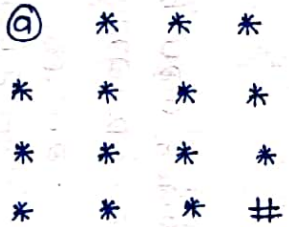
(3)



(4)



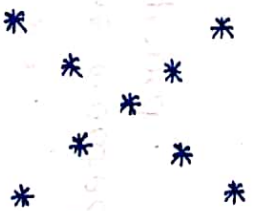
(5)



(6)

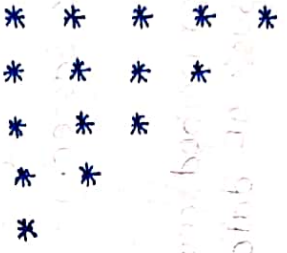
(7)

(8)

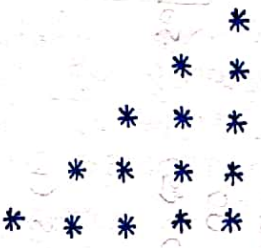


(9)

(10)



(11)

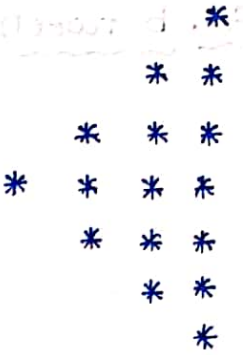


(12)

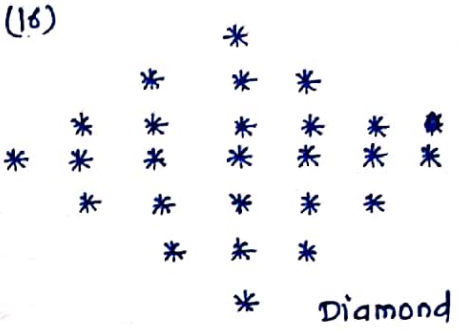
(13)

(14)

(15)



(16)



Diamond

1 to 6 star patterns are in nestle loop.

(7) *(1,1)
- *(2,2)
- - *(3,3)
- - - *(4,4)
- - - - *(5,5)

⇒ rows - 5
columns - 5
no. of operations → (1) star
(2) space
(i==j)

Program:

```
class diagonal {
  main (-) {
For (int i=1 ; i≤5, i++) { // outer for loop for rows.
  For ( int j=1 ; j≤5, j++ ) { // inner for loop for columns
    if (i==j) {
    so print ("*") ;
    }
    else {
     so print ("  ") ;
    }
   }
    soln () ;
  }
  }
```

(8) *(1,1) *(1,5) = 6
*(2,2) *(2,4) = 6 (i+j==6)
*(3,3)
* *(4,4)
* (4,2)
(5,1) *(5,5)
(i==j)

⇒ rows = 5
columns = 5
no. of operations = 2
→ (1) star
→ (2) space

(i==j) &
(i+j==6)

|| → Double pipeline used as 'or'

```
class cross {
 main (-) {
For (int i=1 ; i<=5 ; i++) {
For (int j=1 ; j<=5 ; j++) {
if ((i==j)|| (i+j==6)) {
  sysoprint ("*") ;
}
 else {
  soprint ("  ") ;
 }
 }
  soln () ;
 }
 }
 }
```

(9) * 1 star increasing
* * 2
* * * 3
* * * * 4
* * * * * 5

⇒ Rows – 5 (outer For loop)
columns – 5 (inner For loop)
no. of operations → 1
only increment of star.

```
Class name {
int star = 1 ; // no. of stars present in 1st row.
For (int i=1 ; i≤5 ; i++) {
For (int j=1 ; j≤star ; j++) {
Syso print ("*");
}
Sopln ( ) ;
star ++ ;
}
}
}
```

(10) * * * * * 5
* * * * 4
* * * 3
* * 2 star – –
* 1 no space From left

⇒ rows = 5
columns = 5
no. of operations = 1
decrement star

```
Class name {
int star = 5 ;
For (int i = 1 ; i ≤ 5 ; i++) {
For (int j = 1 ; j ≤ star ; j++) {
Syso print ("*");
}
Sopln ( ) ;
star – – ;
}
}
```

(11) ← spaces decreasing
– – – *
– – – * * ← star increase
– – * * *
– * * * *
* * * * *

⇒ rows = 5
column = 5
no. of operation ⇒ 2
(1) space
(2) star

```
class name {
int space = 4 ; } // 1st row.
int star = 1 ;
For (int i = 1 ; i ≤ 5 ; i++) {
For (int j = 1 ; j ≤ 5 ; j++) {
Syso print (" ");
}
For (int j=1 ; j ≤ star ; j++) {
Syso print ("*");
}
Sopln ( ) ;
space – – ;
star ++ ;
}
}
}
```

(12) * * * * * stars
space increasing — * * * * decreasing
— — * * *
— — — * *
— — — — *

→ Rows = 5
columns = 5
no. of operations ⇒
(1) star
(2) space

```
class name {
  int space = 0 ;
  int star = 5 ;
for (int i=1 ; i≤5 ; i++) {
  for (int j=1 ; j≤5 ; j++) {
   sysop (" ") ;
  }
for (int j=1 ; j≤ star ; j++) {
   syso ("*") ;
}
  sysopln () ;
  star-- ;
  space ++ ;
}
}
```

(13) — * → 1
— — * * * → 3
— * * * * * → 5
* * * * * * * → 7

star value increases by 2

↳ Rows = 4
columns = 7
no. of operation ⇒ 2
① space
② star

```
class name {
   int space = 3 ;
   int star = 1 ;
for (int i=1 ; i≤4 ; i++) {
  for (int j=1 ; j≤space ; j++) {
   syso (" ") ;
  }
for (int j=1 ; j≤star ; j++) {
   syso ("*") ;
  }
   sopln () ;
   star = star+2 ;
   space -- ;
  }
  }
  }
```

(14)

```
*  → 1
* * → 2
* * * – 3
* * * * – 4
* * * – 3
* * – 2
* – 1
```

rows = 7
columns = 4
operation – 1 (star)

```
class name {
    int star = 1
For (int i=1 ; i ≤ 7 ; i++) {
For (int j= 1 ; j ≤ star ; j++) {
  sysop ("*");
}
  sopln (" ");
if (i ≤ 3) {
  star++ ;
}
  else {
  star -- ;
  }
  }
```

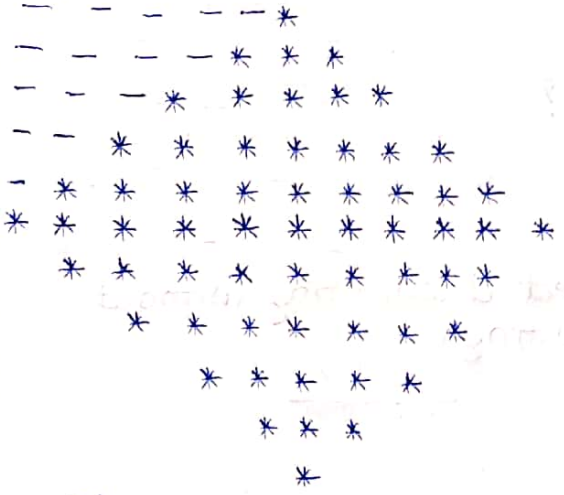
(15)

```
– – – *
– – * *
– * * *
* * * *
– * * *
– – * *
– – – *
```

rows = 7
columns = 4
no. of operations : 2
① star
② space.

```
classname {
    int space = 3 ;
    int star = 1 ;
For ( int i=1 ; i ≤ 7 ; i++) {
For ( int j = 1 ; j ≤ space ; j++) {
    syso ("  ");
}
For (int j=1 ; j ≤ star ; j++) {
    syso ("*");
}
  sopln () ;
if (i ≤ 3) {
  space -- ;
  star ++ ;
}
  else {
   space ++ ;
   star -- ;
  }
  }
  }
```

(16)



Diamond star pattern.

```
int space = 5 ;
int star = 1 ;
for (int i=1; i≤11 ; i++) {
    syso ("  ") ;
}
for (int j=1 ; j≤star ; j++) {
    syso ("*") ;
}
sopln () ;
if (i≤5) {
space -- ;
star = star+2 ;
}
else {
space ++ ;
star = star- 2 ;
}
}
}
```

# EXCEPTION :-

- Exception :- The problem which occurs during runtime is called as "exception".
- Exceptions are always created by JVM, which stops normal flow of the program.
- All Exceptions are present Java.lang.Package.
- Note: The supermost class for exception is "Throwable".

* Difference between Error (compiletime error) & Exception (runtime error):

| Error | Exception |
|---|---|
| (1) These are created by compiler<br>reason: If syntax is wrong. | (1) These are created by JVM.<br>Reason: If JVM faces abnormal situation. |
| (2) These are always showing in Eclipse window. | (2) These are always shown in Console. |

Q] which is the supermost class for exception?
⇒ Throwable.

Q] When we will get Arithmetic Exception?
⇒ Divide by zero.

Q] Tell me any 5 Exceptions in Java?
⇒ (i) classcast Exception (IF we did direct downcasting without upcasting).
(ii) Arithmetic Exception
(iii) NumberFormat Exception
(iv) StringIndex out of Bounds Exception
(v) ArrayIndex OF Bounds Exception.

## * Exception Handling Mechanism:

Exception Handling: Handling the event generated by JVM during program execution is called as "Exception Handling"

- To handle Exception we have three blocks.
(i) Try block
(ii) Catch block
(iii) Finally block

(i) Try block :→
- It is used to declare risky code.
- Every Try block is followed by either catch block or finally block.

For ex:
```
a=10;
b=0;
c=a/b; ⇒ Risky code.
```

(ii) Catch block :→
- It is used to handle the event which is generated by Try block.
- we can declare actual exception with reference variable inside the catch block.
- Always catch block is declared after Try block.
- we can write a infinite no. of catch Blocks for a single try block.

(iii) Finally block :→ It will execute irrespective of Try & catch block. No matter wheather Try, catch block is executed or not But finally will always execute.

For ex: ATM :→ "Thank You For using ATM".

How to handle Exception :-

```
class A {
main (-) {
 int a = 10 ;
 int b = 0 ;
 int c = a/b ; // Risky code.
```

```
 int a = 10 ;
 int b = 0 ;
try {
 int c = a/b ; // risky code
 }
catch (Arithmetic Exception Anky){
syso ("Idiot enter valid
                    denometer) ;
}
syso ("Exception is handled") ;
}
}
```

Checked Exceptions : compiler aware Exceptions are called as checked Exceptions.

Unchecked Exceptions : compiler's unware exceptions are called as Unchecked Exceptions.

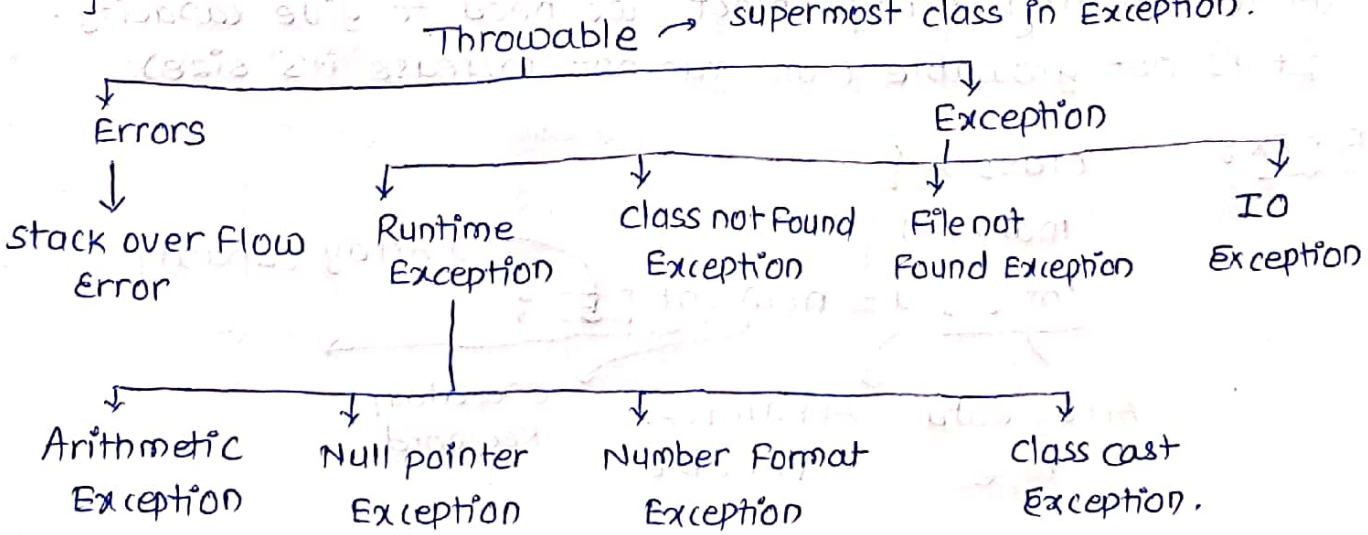
throw : It is used to raise the Exception.

throws : It is used to declare the Exception.

# How to raise exception ?

→

```
class A {
 main (-) {
 syso ("Hi") ;
 syso ("Hello") ;
 throw new ArithmeticException () ;
 }
 }
```



# Difference between throw & throws :-

| throw | throws |
|---|---|
| (1) throw is a keyword which is used to throw on exception forcefully. | (1) throws is a keyword which is used to declare exception which can be generated from method. |
| (2) we can thrown only one exception at a time. | (2) we can declare multiple exception at a time. |
| (3) Throw keyword is used inside the method body | (3) Throws keyword is used in method ~~declration~~ declaration. |
| (4) Throw keyword throw only only runtime exception | (4) Throws keyword declared the runtime and compiler exception. |
| (5) Throw keyword throw an exception by creating the object of exception class. | (5) Throws keyword declare an exception by using class name only. |

# #Arrays :-

variable → used to stored the single data

Arrays → used to stored the multiple data of same data type (Homogeneous data)

Collection → used to stored the multiple data of different type. (Heterogeneous data)

Array : It is data structure in which we can store multiple data of same type.

- It is homogeneous in nature.
- while creating Array object, we need to give capacity.
- It is not growable (we can not increase it's size).

For Ex :

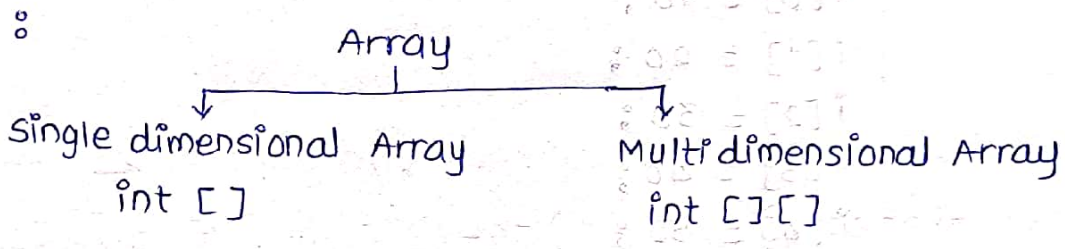
```
class A {
main (-) {
int [] i = new int [5] ;
```

- `int []` : Array data type
- `i` : Array ref. variable
- `new` : Operator / Keyword
- `[5]` : Array capacity

Program:

```
public class B { // single dimensional array
 main (-) {
String [] s = new string [5] ;
 s[0] = "Ankush" ;
 s[1] = "Jeevan" ;
 s[2] = "Kiran" ;
 s[3] = "Kapil" ;
 s[4] = "Mohit" ;
→ // To print entire data
 Syso (s[5]) ;
 }
}
```

## Array :




## Multi-dimensional Array :→

- Multi-dimensional array is used for matrix representation.

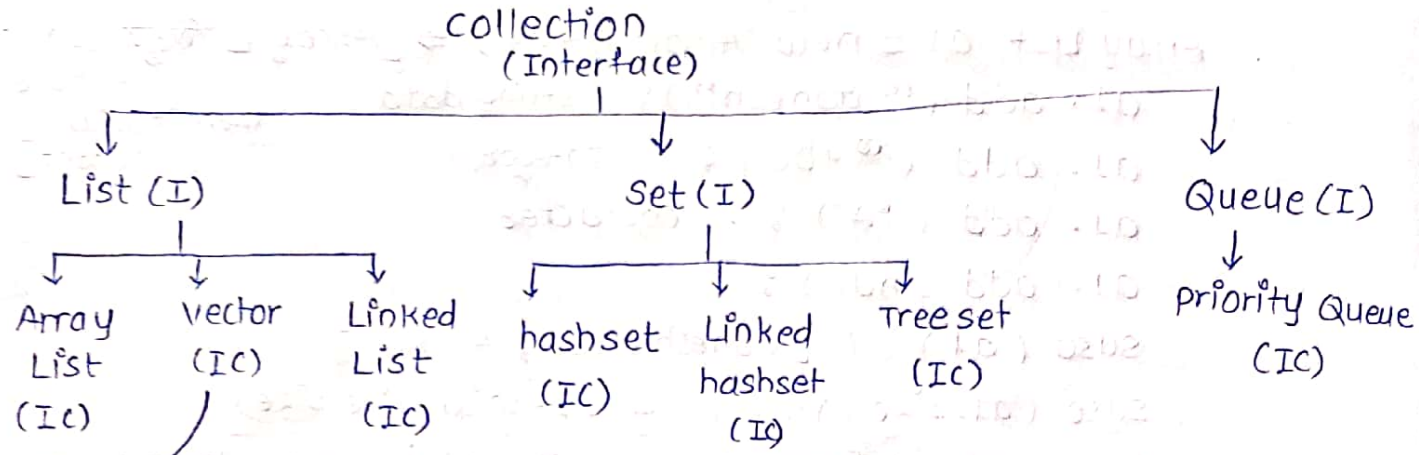
E.g. $\begin{bmatrix} 10 \ (0,0) & 20 \ (0,1) \\ 30 \ (1,0) & 40 \ (1,1) \end{bmatrix}$

Program :→

```
package Array ;
// For Matrix Representation
Public class Multidimensional_array {
 main (-) {
 int ar[][] = new int[2][2] ;
  ar[0][0] = 10 ;
  ar[0][1] = 20 ;
  ar[1][0] = 30 ;
  ar[1][1] = 40 ;
// syso (ar[1][1]) ; // 40
// syso (ar[0][0]) ; // 10

// To print entire matrix
For (int i=0 ; i<=1 ; i++){// outer loop
  For (int j= 0 ; j<=1 ; j++) {// inner loop
syso(ar[i][j] + "   ") ;
}
 sysopln ( ) ;
}
}
}
```

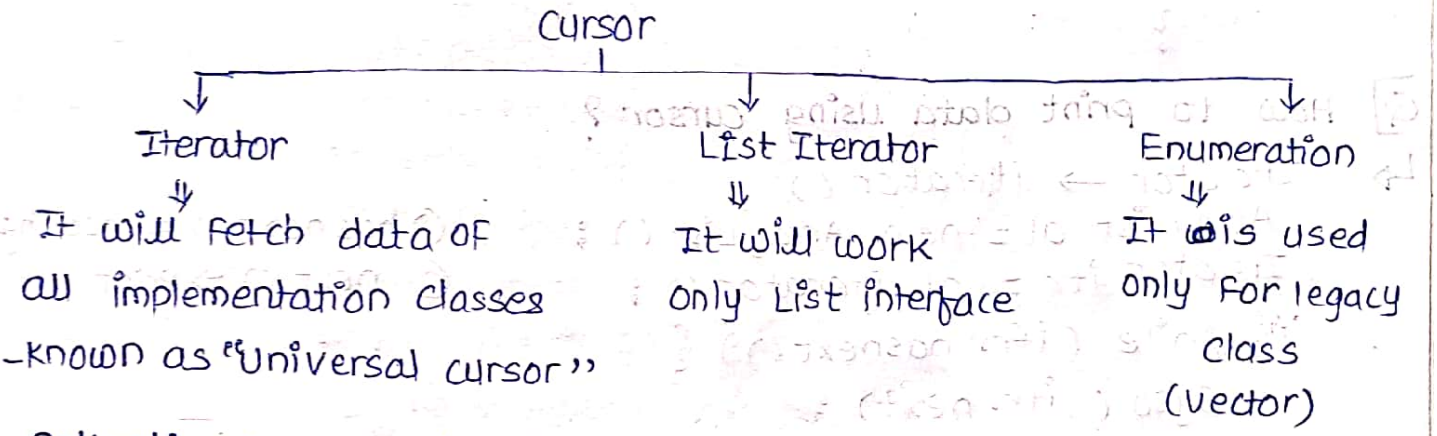
# Collection :-




It is a legacy class in Collection (oldest)

IC - implementation class
I - Interface.

- Cursor → To fetch multiple data




Iterator ⇓ It will fetch data of all implementation classes -known as "Universal cursor"

List Iterator ⇓ It will work only List interface

Enumeration ⇓ It is used only for legacy class (vector)

**Collection :-** It is the ready made framework (Interface) which is subdivided into 3 - sub interfaces.

Features : (i) It will accept heterogeneous data
(ii) size of collection is growable (we can change this)
(iii) Disadvantages of array removed by collection.

Program :- class arraylist {
main(-) {
import java.util.ArrayList;

```
ArrayList al = new ArrayList();  => [Array L लिहून control +space then it will import]
al.add("Ganesh"); // string data
al.add(100); // Integer
al.add('A'); // character
al.add(null);
syso(al); // [Ganesh, 100, A, null]
syso(al.size()); // 4 (It will give size)
syso(al.isEmpty()); // It will check wheather the collection is empty or not.
syso(al.contains(100)); // True.
syso(al.get(0)); // To get data from perticular Index.
}
}
```

Q] How to print data using cursor ?

↳ Iterator → iterator()

```
ArrayList al = new ArrayList();
Iterator itr = al.iterator();
while(itr.hasnext()){
syso(itr.next);
}
```

① hasnext() ⇒ checking
② next() ⇒ print

* For each loop :- It is advanced version of for loop.

syntax : For (Object s1 : al) {

Object is data type

entire data

Syso(s1); // [Ganesh, 100, A, null]

(All data is printed)

Q] How to fetch data by using Enumeration?

↳ Enumeration en = V.elements();

while (en.hasmoreelements()) {
syso (en.nextelements());
}

(i) hasmoreelements();
(ii) nextelement();

Program:-

```
Package collection;
importjava.util.ArrayList;
importjava.util.Iterator;
importjava.util.ListIterator;
Public class Array_List {
  main (-) {
ArrayList al = new ArrayList();
   al.add ("Ganesh");
   al.add (100);
   al.add ('A');
   al.add (null);
→ // To print entire data
  // syso (al);
→ // To get the size
  // syso (al.size()); // 4
→ // To check empty or not
  // syso (al.isEmpty()); // False or ~~true~~
→ // containing data or not
  // syso (al.contains (200)); // False
→ // To get perticular data from perticular Index.
  // syso (al.get (0)); // Ganesh
→ // To fetch data is using Iterator (Universal cursor)
  // Iterator itr = al.Iterator();
  // while (itr.hasNext()) {
  // syso (itr.next());
  // }
→ // To fetch data using List Iterator
  // ListIterator litr = al.ListIterator();
  // while (litr.hasNext()){
  // syso (litr.next());
  // }
```

```
// To print data using For each loop
For (object S : al){
syso (S);
}
}
}.
```

# Collection(I)

## List(I)

(1) Array List (Implementation class)
- Duplicates are allowed.
- Allows any no. of null values.
- Order of insertion is maintained.
- Default capacity is 10.
- Best choice: Retrieval operation
- worst choice: Manipulation operation.

(2) Vector (Implementation class)
- Duplicates are allowed
- Allow any no. of null values.
- order of insertion is maintained.
- Default capacity is 10.
- Best choice: Retrieval operation.
- worst choice: Manipulation operation.

(3) LinkedList (IC)
- Duplicates are allowed.
- Allow any no. of null values.
- Order of insertion is maintained
- NO default capacity
- Best choice: Manipulation operation.
- Worst choice: Retrieval operation.

## Set(I)

(1) Hashset (IC)
- Doesn't allow duplicates
- Allow only one value
- order of insertion is Random.
- There is no default capacity.
- Best choice: It will remove duplicate element when order of insertion is not mandatory.
- Data structure is hashable.

(2) Linked Hashset
- Does not allow duplicates
- Allow only one null value
- order of insertion is maintained.
- There is no default capacity.
- Best choice: To remove duplicates; order of insert is mandatory.
- Data structure is hybrid (Linear + Hashable)

(3) TreeSet
- Doesnot allow duplicate
- Null value is not allowed
- order of insertion is in ascending order.
- There is no default capacity.
- Best choice: Remove duplicate element when order of insertion is in ascending order.
- Data structure is balanced Tree (Hashable)

Program :

```
Package collection ;
import java.util.Enumeration ;
import java.util.Iterator ;
import java.util.ListIterator ;
import java.util.Vector ;
Public class Vectors {
Public static void main (String [] args) {
    Vector V = new Vector () ;
    V.add ("Sai Pallavi") ;
    V.add (100) ;
    V.add ('A') ;
    V.add (null) ;
↳ // To print entire data
   // syso (V) ;
↳ // To check empty or not
   // syso (V.isEmpty ()) ; // false
↳ // To get size of collection
   // syso (V.size ()) ; // 4
↳ // To check contains data or not
   // syso (V.contains (200)) ; // false
↳ // To get Index
   // syso (V.get (3)) ; // null
↳ // By using Iterator
   // Iterator itr = V.Iterator () ;
   // while (itr.hasNext ()) {
   // syso (itr.next ()) ;
   // }
↳ // Enumeration ListIterator
   // ListIterator litr = V.listIterator () ;
   // while (litr.hasNext ()) {
   // syso (litr.next ()) ;
   // }
↳ // Enumeration (used for vector)
   // Enumeration en = V.elements () ;
   // while (en.hasMoreElements ()) {
   // syso (en.nextElement ()) ;
   // }
↳ // Using For loop
   for (object a1 : V) {
   syso (a1) ;
}
}
}
```

Program :-

```
package collection ;
import java.util. Iterator ;
import java.util.LinkedList ;
import java.util.ListIterator ;
Public class Linked_list{
main (-) {
LinkedList ll = new LinkedList() ;
ll. add ("stallin") ;
ll. add (200) ;
ll. add ('A') ;
ll. add (null) ;
→ // To print entire data
  // syso (ll) ;
→ // To get size of collection
  // syso (ll.size()) ;
→ // To check contains
  // syso (ll. contains (200)) ; // o/p True
→ // To check empty or not
  // syso (ll. isEmpty ()) ;
→ // To fetch data by using Iterator
  // Iterator itr = ll. iterator () ;
  // while (itr. hasNext ()) {
  // syso (itr. next ()) ;
  // }
→ // By using ListIterator
  // ListIterator litr = ll. listIterator () ;
  // while (litr. has Next ()) {
  // syso (litr. next ()) ;
  // }
→ // Fetch using for each loop
  For (object a : ll) {
  syso (a) ;
}
}
}
```

Program :-

```
Package collection ;
import java.util.Hashset
import java.util.Iterator
Public class Hash-set {
main (-) {
HashSet hs = new HashSet() ;
hs.add ("Kiran") ;
hs.add ("500") ;
hs.add ('A') ;
hs.add (null) ;
// To print entire data
// syso (hs) ;  ⟹ o/p : [null, 1, S, Surekha]
// To check empty or not
// syso (hs.isEmpty()) ;
// To fetch data using Iterator
// Iterator itr = hs.iterator() ;  ⇒ only iterator cursor can used no used of List Iterator possible and No Enumeration.
// while (itr.hasNext()) {
// syso (itr.next()) ;
// }
// To fetch data using For each loop.
For (object z : hs) {
syso (z) ;  ⟶ o/p ⇒ null
}                   1   ← (printing from bottom to top)
}                   S
}                   Surekha
```

Program :-

```
Package collection ;
import java.util. Iterator ;
import java.util. TreeSet ;
Public class treeset { // it is exactly like array
                        (the data should be same)
Public static void main (String [] args) {
    TreeSet ts = new TreeSet () ;
     ts. add ("Kiran") ;
     ts. add ("Kunnu") ;
     ts. add ("Kanba") ;
     ts. add ("Karan") ;
// To print entire data
// syso (ts) ;

// By using cursor
// Iterator itr = ts. iterator () ;
// while (itr. hasNext ()) {
// syso (itr. next ()) ;
// }
// For each loop
   For (Object i : ts) {
   syso (i) ;
}
}
}
```

Program :-

```
Package collection ;
import java. util. Iterator ;
import java . util . Linked Hash set ;
public class linked_ hash set {
public static void main (string [] args) {
  LinkedHashSet lhs = new LinkedHashSet () ;
  lhs. add ("Kiran") ;
  lhs. add (17) ;
  lhs. add ('B') ;
  lhs. add (null) ;
// To print entire data
// syso (lhs) ;
// To check empty or not
// syso ( lhs. is Empty ()) ;
// To check data contains or not
// syso ( lhs. contain (100)) ; // False
// To get size OF collection
// syso (lhs. size ()) ;
// To Fetch data using Iterator
// Iterator itr = lhs. iterator () ;
// while (itr. hasNext ()) {
// syso ( itr. next ()) ;
// }
// For each loop
  For (object O : lhs) {
   syso (O) ;
}
}
}
```

# Map(I) :→ There are two implementation classes in Map.

(i) Hashmap (IC)

(ii) Hashtable (IC)

- Map is the collection.
- Here data can be stored in the form of keyword and values format (Key, value).

For Example :→

| Key | values |
|---|---|
| 101 | "Hrithik Roshan" |
| 102 | "Vijay Deverkonda" |
| 103 | "Allu Arjun" |
| 104 | "Thalpathy Vijay" |
| 105 | "Surya Sivakumar" |

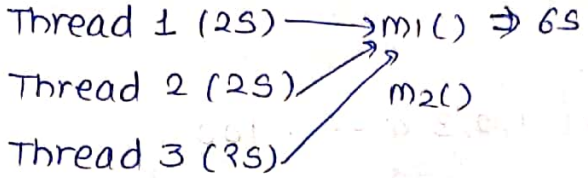
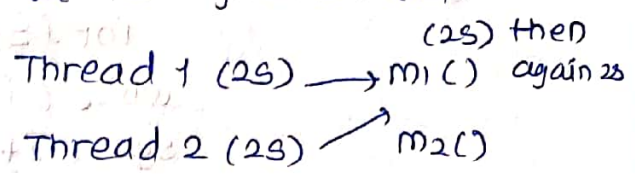
syso ( )

⇒ Format [ — — — — ] ← O/P

Program :→

```
Class A {
main (-) {
Hashtable < Integer, String> ht = new Hashtable < Integer,
                                                  String> ();
ht. put (101, "Hrithik Roshan"); // Entry 1
ht. put (102, "Vijay Deverkonda"); // Entry 2
ht. put (103, "Allu Arjun"); // Entry 3
ht. put (104, "Thalpathy Vijay"); // Entry 4
ht. put (105, "Surya Sivakumar"); // Entry 5
// print Entry 1
syso (ht. Keyset); // 101, 102, 103, 104, 105
// contain ()
syso (ht. containkey (106)); // false
// size ()
syso (ht. size ()); // 5
// Remove key
ht. remove (101);
syso (ht); // 102, 103, 104, 105
// contains ()
Syso (ht. Contain (101)); // false
```

# Difference between Hashmap and Hashtable :-

| Hashmap | Hashtable |
|---|---|
| (i) It is Non-synchronized.   (multiple threads can be accessed at a time) | (i) It is synchronized.   (Two threads can not be accessible at the same time) |
| (ii) It is not Threadsafe | (ii) It is Threadsafe. |
| (iii) performance is faster | (iii) performance is slower. |
| (iv) It will accept only one key as null But it will accept multiple null values. | (iv) It will not allow even single key as null and also value as null. |
| (v) It is new form which came from JDK 1.2 onwards. | (v) It is a legacy (oldest). |

# Wrapper classes :-

Q]. can Java is 100% object oriented ?

Ans: No, because it supports primitive and Non-primitive data types.

| Data types | Wrapper class |
|---|---|
| byte → | Byte |
| short → | Short |
| int → | Integer (~~parseInt~~) (ParseInt()) |
| long → | Long |
| double → | Double |
| float → | Float |
| char → | Character |
| boolean → | Boolean |

- By Using wrapper classes, we are converting primitive data types to non-primitive data types so that our Java becomes 100% object oriented.

# Logical Programms :-

(1) Print numbers From 1 to 100 ??

⇒ Using while loop :

```
int i=1 ;
while ( i≤100) {
syso ( i++) ;  // 1,2,3,4 ---- 100
}
```

(2) → By using For loop :

```
For (int i= 1 ; i≤ 100; i++) {
 syso (i) ;  // 1, 2, 3, 4 --- 100 .
}
```

(3) Print characters From A to Z .

⇒

```
For ( char ch='A' ; ch ≤'Z' ; ch++) {
 syso (ch) ; // A, B, C, D ------ Z.
}
```

(4) Print characters from Z to A .

⇒

```
For ( char ch ='Z' ; ch >='A' ; ch--) {
 syso (ch) ; // Z, Y, X, W ---- A
```

(5) write a Factorial of 5 ??

$5! = 5 \times 4 \times 3 \times 2 \times 1$

⇒

```
int num =5 ;
int Fact = 1 ;
```

```
Fact = 1 ;  —(1)
Fact = fact * 2 ;
 (2)     (1) * (2)  —(2)
Fact = Fact * 3
 (6)     (2) * (3)  —(3)
Fact = Fact * 4
 (24)    (6)  (4)   —(4)
Fact = fact * 5  —(5)
 (120)   (24) * (5)
```

⇒ smart work

```
Fact = Fact * i
```

↓ Factorial

Example:

```
class A {
  main (-) {
    int num = 5 ;
    int fact = 1 ;
  For (int i = 1 ; i ≤ num; i++) {
      fact = fact * i ;
    }
  syso (Fact) ; // 120.
```

(6) Print multiplication of numbers from 1 to 10.

↳

```
int num = 10 ;
int Prod = 1 ;
 Prod = 1 ;
Prod = prod * (2)
 (2)    (1)
Prod = Prod x (3)
 (6)    (2)
Prod = Prod x (4)
 (24)   (6)
Prod = Prod x (5)
 (120)  (24)
  !
  !
Prod = Prod x (10)
 ( )
```

? = 1x2x3x4x5x6x7x8x9x10

Short logic (smart logic)

```
Prod = Prod * i
```

Example:

```
class Multiplication {
   main (-) {
   int num = 10 ;
   int Prod = 1 ;
 for (int i = 1 ; i ≤ num ; i++) {
      Prod = Prod * i ;
    }
  syso (Prod) ;
  }
```

(7) Addition of numbers From 1 to 10.

↳
```
int num = 10;
int sum = 0;
sum = 1;
sum = sum + (2)
(3)
sum = sum + (3)
(6)      (3)
sum = sum + (4)
  :
  :
sum = sum + (10)
```

⇒ smart logic

```
Sum = sum + i
```

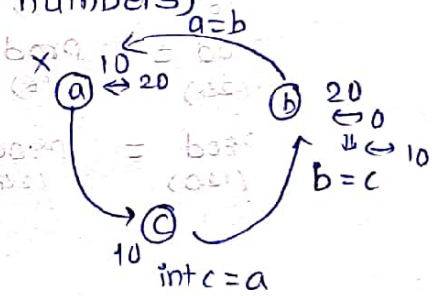
For Example

```
class A {
main(-) {
int num = 10;
int sum = 0;
For (int i = 1 ; i ≤ 10 ; i++) {
    sum = sum + i;
}
syso (sum);
}
}
```

(8) Swapping of numbers (interchange of the numbers)

⇒

```
int a = 10;
int b = 20;
int c = a;
a = b ; //
b = c ; //   } variable intialization.
syso (a) ; //20
syso (b) ; //10
```




(9) Reverse the string :

```
String org = "Ankush";
String rev = "";
for(int i = org.length()-1 ; i >= 0 ; i--) {
    rev = rev + org.charAt(i);
}
syso (rev) ;   // hsuknA
}
```

Ankush ⇒ length = 6
012345 → Starting From zero. (0 to 5)

(10) String Palindrome

Palindrome : madam $\overset{\text{Reversed}}{\leftrightarrow}$ madam.

Ex: rotator, level, MOM, DAD, WOW

Program:

```
String org = "Ankush";
String rev = "";
For (int i = org.length()-1 ; i >= 0 ; i--) {
    rev = rev + org.charAt(i) ;
}
syso (rev); // hSUKnA
if (org.equals(rev)) {
syso ("Given string is palindrome");
}
else {
syso ("Given string is not palindrome");
}
}
```

(11) Reverse the number :

```
int num = 1234 ;
int rev = 0 ;
For (int i = num ; i > 0 ; i = i/10) {
    int rem = i % 10 ;
    rev = rev * 10 + rem;
}
syso (rev); //4321
}
```

| % | / |
|---|---|
| ⇓ | ⇓ |
| reminder | quotient |

(12) Number palindrome :⇒ 101 $\overset{\text{rev}}{\leftrightarrow}$ 101

```
int num = 101 ;
int rev = 0 ;
For (int i = num ; i > 0 ; i = i/10) {
    rev = rev * 10 + rem ;
}
    syso (rev); // 101
} if (num == rev) {
syso ("Given no. is palindrome");
} else {
    syso ("Given no. is not palindrome"); }
```

(13) Even and odd numbers:

```
Class A {
 main (-) {
  int num = 2 ;
 if ( num % 2 == 0 ) {
  syso (" Given number is even");
  }
  else {
  syso (" Given number is odd") ;
  }
```

num % 2 == 0

```
   1
2 ) 2
  - 2
  ---
    0
    ↓
Reminder
(Then even)
```

```
   1
2 ) 3
  - 2
  ---
    1
    ↳
Reminder
So odd
```

(14) Prime numbers : (Divide by 1 and itself)

only 2 times divide

```
Class A {
main (-) {
 int num = 8 ;
 int count = 0 ;
 for (int i = 1 ; i ≤ num ; i++) {
   if (num % i == 0) {
     count++ ;
   }
   if (count == 2) {
    syso(" Given number is prime number") ;
   }
   else {
   syso (" Given number is not prime") ;
   }
```

(15) Remove special single character :

```
class A {
main (-) {
String s = " Vai@bhav";
String correctName = s.replace("@", " ");
syso ( correctName) ;
}
}
```

(16) Remove multiple special characters :

```
class A {
main (-) {
String s = " Vai#bh@v";
String remove = s.replaceAll ("[^a-z A-Z], " ");
System. out . println (remove) ;
}
}
```

(17) Counting white spaces :-

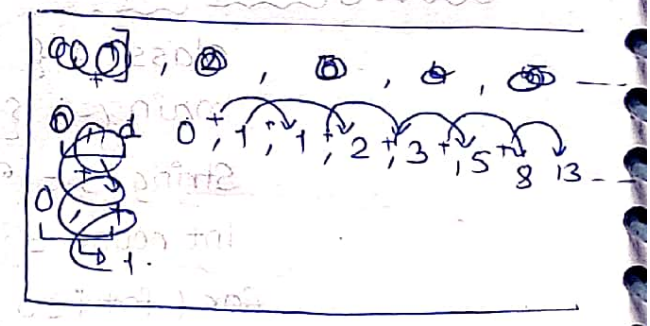
```
class A{
main (-) {
String S = " V E L O C I T Y";
int count = 0 ;
for ( int i=0 , i <= name .length ()-1 ; i++) {
char actChar = S. charAt (i);
if ( actChar == ' ') {
count ++ ;
}
}
syso ( count) ;
}
}
```

(18) Remove white spaces :

```
class A {
 main (-) {
String S = " V E L O C I T Y ";
String actN = S. replaceAll (" ", "") ;
                              ↳ space
Syso ( actN) ;
}
}
```

(19) Fibonacci Series :

```
class A {
main (-) {
 int n1 = 0 ;
 int n2 = 1 ;
 int sum = 0 ;
 sysp ( n1 + " ") ;
 sysp ( n2) ;
For (int i=2 ; i<=10 ; i++) {
   sum = n1 + n2 ; // 0+1 = 1
 sysp(" " + sum) ;
    n1 = n2 ;
    n2 = sum ;
 }
 }
 }
```



(20) Armstrong number :▷ Armstrong no = 153 = $(1)^3 + (5)^3 + (3)^2$
1 + 125 + 9

```
class A {
main (-) {
int num = 153 ;
int actN = num ;
int r = 0 ;
int armN = 0 ;
while (num > 0) {
   r = num % 10 ;
  armN = r*r*r + armN ;
    num = num / 10 ;
}
 IF (actN == armN) {
 syso ( armN + " " + " Given no is Arm-
                          strong") ;
}
else {
syso ( armN + " " + " Given no. is not
                          Armstrong") ;
}
}
}
```